“Comparació d'aproximació genètica i aproximació per aprenentatge per reforç
en un entorn de simulació per a robots netejadors”

Estudiant: Marina Gispert i Muñoz

Index

1. Resum

En aquesta tesi es pretén abordar diverses solucions de navegadors per a robots netejadors. La motivació prové dels robots de neteja casolans, que tendeixen a oferir poca eficàcia. L'objectiu és millorar l'eficàcia d'aquests robots. Per a millorar l'eficàcia del robot, es planteja buscar nous sistemes de navegació que incrementin la superfície escombrada.

La solució proposada consistirà en la creació d'un conjunt de controladors mitjançant algorismes genètics i aprenentatge per reforç que obtinguin navegadors que millorin l'eficàcia de l'algorisme actual.

La metodologia seguida per a mesurar l'eficàcia de un algorisme consisteix en l'ús d'un simulador on ser avaluat donades unes condicions de temps i entorn.

En la primera part del projecte, es crea una eina simulador nova per motius de velocitat i eficiència.

En una segona part del projecte, es desenvolupa un sistema genètic amb diversos tipus de genètics probabilístics i un sistema d'aprenentatge per reforç, així com un navegador aleatori de referència. En el cas de l'aprenentatge per reforç, es presenta un sistema novedós que permetria el robot aprendre en una casa mentre opera.

Finalment s'han executat, avaluat i contrastat els controladors obtingudes i s'han extret conclusions.

2. Motivació

Avui en dia és comú i fins i tot normal trobar-se amb aspiradors automàtics amb la missió de netejar la casa.

Aquesta neteja hauria de ser el més eficient possible, tanmateix, els algorismes que exhibeixen aquests robots manquen d'un intel·ligència prou definida i necessiten varies hores per a un simple escombrat.

Aquest projecte es basa en l'observació del robot netejador ASTAN model BL001A. Aquest model té un comportament d'eficiència comparable als altres mercat i té l'avantatge que en el manual d'instruccions es detalla l'algorisme que segueix.

Decidim que aquests algorismes es poden millorar i optem a explorar algorismes d'autoaprenentatge. Definim el projecte com la realització de controladors de referència i altres controladors creats a partir d'algorismes genètics i creats a partir d'aprenentatge per reforç. Aquests controladors seran finalment comparats en termes d'eficàcia i temps.

A més, després d'un estudi de possibilitats inclòs en la memòria, s'estableix que es crearà un simulador de l'entorn i el robot per a possibilitat la creació dels controladors i el seu testeig.

Finalment, aquest projecte ha acabat conformant 4 parts diferenciades, tal i com es presenta a continuació.

3. Objectius

L'objectiu més general del projecte és aconseguir un controlador per a robots aspirador com l'ASTAN model BL001A (tipus Roomba).

A partir d'aquest objectiu general, es perfilen quatre objectius específics. Aquests objectius són:

- *Desenvolupar una eina de simulació per a la navegació del robot*. Aquest objectiu comprèn la creació d'un entorn de simulació programat per a la navegació del robot i permetre l'avaluació de la seva navegació. Aquesta eina desenvolupada haurà d'acomplir els requeriments que queden descrits més endavant a l'apartat Requisits del simulador.
- *Desenvolupar un controlador basat en computació evolutiva*. Aquest objectiu comprèn la creació d'un controlador del robot dins l'entorn de simulació creat. Aquest controlador ha de ser el més eficaç resultant de tot el procés de selecció per computació evolutiva segons els bancs de proves descrits en el mateix apartat.
- *Desenvolupar un controlador basat en aprenentatge per reforç*. Aquest objectiu comprèn la creació d'un controlador del robot dins l'entorn de simulació creat.
- *Comparació dels controladors obtinguts*. Aquest objectiu comprèn l'exposició clara i objectiva dels resultats de comparar els tipus de controladors principals (obtinguts per computació evolutiva i aprenentatge per reforç) amb els controladors auxiliars de validació i entre ells. Amb aquest objectiu es pretén extreure conclusions objectives sobre ambdós controladors.

4. Introducció

En aquesta introducció es mostrarà les bases teòriques de la computació evolutiva i l'aprenentatge per reforç, es mostrarà el robot origen i model d'aquest projecte, es mostraran resumides algunes de les eines de disseny i simulació disponibles i finalment s'enumeraran alguns robots adients per a implementar les solucions obtingudes.

4.1. Sistemes de navegació

En aquest capítol es mostren les bases teòriques dels algorismes per computació evolutiva, les bases de l'algorisme d'aprenentatge per reforç i les característiques tècniques que hauran de reproduir els models de robot.

4.1.1. Computació evolutiva

La computació evolutiva és una branca de la intel·ligència artificial que involucra problemes d'optimització combinatòria inspirant-se en els mecanismes d'evolució biològica.

Existeixen tres grans branques de la computació evolutiva: la programació evolutiva, les estratègies evolutives i els algorismes genètics.

La programació evolutiva es basa en l'evolució de màquines d'estats finites. És pot considerar com una variació dels algorismes genètics en la qual els individus estan representats per estats d'un autòmat finit.

Les estratègies evolutives tenen com a objectiu l'optimització de paràmetres en una funció. Per tant serveixen quan donada una funció, es tracta de seleccionar l'individu (valors dels paràmetres) més òptim per a aquesta funció.

En aquest projecte s'ha escollit el mètode dels algorismes genètics com a mètode per a crear el controlador amb un algorisme per computació evolutiva.

4.1.1.1. Algorismes genètics

Els algorismes genètics són algorismes de solució de problemes plantejats a base de avaluar i modificar (per creuament i mutació) a través d'iteracions (anomenades generacions) els candidats a solució, anomenats individus.

El funcionament bàsic consisteix la definició de l'estructura dels individus de la població inicial (definir quines característiques representen els gens) i inicialitzar la població, preferiblement aleatòriament.

Iterativament, es sotmeten els individus de la població a avaluació i se seleccionen els millors individus. Aquests individus seleccionats són usats per a crear la nova generació amb els mètodes escollits (creuament i/o mutació). Després es reemplacen part de la població antiga per població nova.

4.1.2. Aprenentatge per reforç

L'aprenentatge per reforç (Reinforcement Learning) és una tècnica d'entrenament d'algorismes basada en el concepte acció-recompensa.

Un model bàsic de representació d'un problema per aprenentatge per reforç conté: un espai d'estats *S*, un espai d'accions *A*, un conjunt de regles de transició entre estats, unes regles que determinin l'escalar

recompensa per a cada transició d'estats i regles que determinen l'estat actual a través de l'observació del seu entorn.

4.1.2.1. Problemes de decisió de Markov

Els models basats en aprenentatge per reforç en màquines són modelats normalment a través de problemes de decisió de Markov. Podem descriure un problema de Markov com una tupla $(\mathcal{X},\mathcal{A},P,r,\gamma)$ on $\mathcal{X}$ és l'espai d'estats, $\mathcal{A}$ és l'espai d'accions, P són les probabilitats de transicions, r és la funció recompensa i γ és el factor de descompte usat ($0 \leq \gamma \leq 1$).

Si definim els espais $\mathcal{X}$ i $\mathcal{A}$ com a discrets, podem descriure la probabilitat de passar a l'estat X(t+1) des de l'estat X(t) quan es dona l'acció a t A(t) com:

$$\mathbb{P}[X(t+1) = y \mid X(t)i = x, A(t) = a] = P(x,a,y). \quad \textbf{(1)}$$

I l'esperança de la possible recompensa es pot descriure com:

$$E[R(t) \mid X(t) = x, A(t) = a\ ] = r\,(x,a). \quad \textbf{(2)}$$

L'objectiu del robot és maximitzar la suma total de recompenses (amb el descompte γ triat):

$$E\left[\sum_t \gamma^t R(t)\right] \quad \textbf{(3)}$$

Una política és aquella funció π: $\mathcal{X} \longrightarrow \mathcal{A}$ que assigna a cada estat $x \in \mathcal{X}$, una acció $a = \pi(x)$ que el robot haurà d'executar sempre que es trobi a l'estat x. El valor d'una política és l'esperança total de recompensa que el robot rebrà seguint aquesta política. Matemàticament s'expressa amb la funció:

$$V^{\pi}(x) = E\left[\sum_t \gamma^t R(t) \,\middle|\, X(0) = x, A(t) = \pi(X(t))\right] \quad \textbf{(4)}$$

La política òptima π^*, és la política que té el major valor a tots els estats $x \in \mathcal{X}$: $V^*(x) \geq V^{\pi}(x)$.Usant-la en la definició de V^{π}, podem obtenir la formula de calcul de V^{π} recursiva:

$$V^{k+1}(x) = r(x,\pi(x)) + \gamma \sum_y P(x,\pi(x),y) V^k(y) \quad \textbf{(5)}$$

Si definim la funció Q :

$$Q^{\pi}(x,a) = r(x,a) + \gamma \sum_y P(x,a,y) V^{\pi}(y) \quad \textbf{(6)}$$

Per a una π^* escriuríem :

$$Q^*(x,a) = r(x,a) + \gamma \sum_y P(x,a,y) V^*(y) \quad \text{on} \quad \left\{ \begin{array}{l} V^*(x) = \max_a Q^*(x,a) \\ \pi^*(x) = \arg\max_a Q^*(x,a) \end{array} \right\} \quad \textbf{(7)}$$

Reemplaçant V* i iterant la funció Q*, obtenim la formula recursiva de Q*:

$$Q^{k+1}(x,a) = r(x,a) + \gamma \sum_y P(x,a,y) \max_b Q^k(y,b) \quad \textbf{(8)}$$

que permet calcular iterativament Q* a partir d'una Q inicial i un coeficient de descompte γ establert.

4.1.2.2. Aprenentatge per diferencia temporal

Amb cada nova iteració, s'obté més informació. La nova informació ha de ser afegida al coneixement previ, ponderada per una serie $\{ \alpha_t \}$ que són els coeficients de la llargada del pas ($0 \leq \alpha \leq 1$). Modificant la equació 5 i la equació 8 afegint α, obtenim :

$$V^{(t+1)}(x_t) = V^{(t)}(x_t) + \alpha_t (r_t + \gamma V^{(t)}(x_{t+1}) - V^{(t)}(x_t)) \quad \textbf{(9)}$$

$$Q^{(t+1)}(x_t, a_t) = Q^{(t)}(x_t, a_t) + \alpha_t \left(r_t + \gamma \max_b Q^{(t)}(x_{t+1}, b) - Q^{(t)}(x_t, a_t) \right) \quad \textbf{(10)}$$

On escollint una γ adient i una serie $\{ \alpha_t \}$, es pot calcular iterativament V^{π} i Q*. Usualment, s'itera fins que la distància entre la matriu d'iteració *n* i *n+1* és menor a un escalar establert prèviament.

4.1.3. ASTAN model BL001A

El robot ASTAN model BL001A afirma en el seu manual d'ús que usa un recorregut hexagonal incrementant poc a poc el costat de l'hexagon.

Imatge 1: Imatge del robot ASTAN model BL001A

Al detectar una paret física o paret virtual les recorre al llarg fins a detectar una altra paret per a calcular el total de la superfície a netejar.

Al detectar un obstacle gira sobre sí mateix reajustant el recorregut.

El robot té tres programes seleccionables, segons la mida de l'habitació a netejar:

S	10-15 m^2
M	15-25 m^2
L	Més gran que 25 m^2

Taula 1: Programes de l'ASTAN BL001A

4.2. Simuladors disponibles al mercat

En aquest apartat es fa un breu resum dels simuladors més famosos i usats actualment.

Inicialment, ha calgut fer una primera aproximació als simuladors del mercat i s'ha fet un estudi dels tres més significatius: Marilou, MS Robotics i Webots. Es pot observar les seves característiques a la taula següent

Nom	AnyKode Marilou	MS Robotics	Webots
Prestacions	Disseny i simulació	Disseny i simulació	Disseny i simulació
Requeriments d'ús	C/C++,C++ CLI, C#, J#, VB#, amb els compiladors MS Visual Studio, DevC++, Borland C++ RAD Studio i G++ for Linux.	Microsoft Visual Studio o Microsoft Visual C#	C, C++, Java, Python, Matlab o URBI
Llicència	Educativa - 0€	Gratuïta	Educativa - 246 €

Taula 2: Taula de característiques

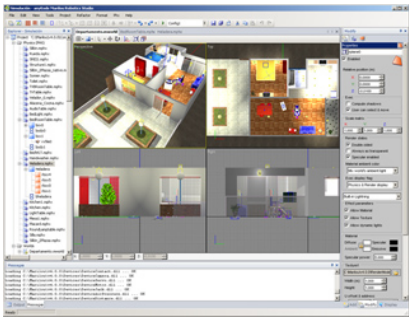
4.2.1. AnyKode Marilou

Aquest entorn de disseny i simulació de robots és usat per la industria i centres de recerca creada per l'empresa Anykode [5] .

És una eina potent per als càlculs complexos (lleis físiques, forces, parells, interacció amb superfícies). Un altre punt fort és el fet de ser desenvolupat per una empresa, la qual cosa aporta fiabilitat en el producte i garanties de solvència.

A través del seu SDK MODA (Marilou Open Devices Access), es pot usar llibreries per a la programació en C/C++,C++ CLI, C#, J#, VB#, amb els compiladors MS Visual Studio, DevC++, Borland C++ RAD Studio i G++ for Linux. La llicència educativa és gratuïta.

Aquesta eina està pensada per al seu ús en el modelatge de sistemes complexos, amb grans requeriments de càlcul i amb simulació de les vistes resultants, amb gran complexitat de càlcul.



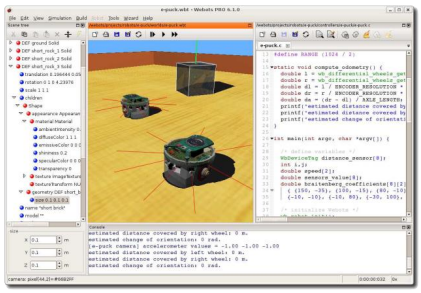
Imatge 2: Exemple d'ús de Marilou

4.2.2. Webots

Webots és un plataforma de desenvolupament creada per l'empresa Cyberbotics [6] . És una plataforma completa per a modelar, simular i programar robots.

Aquesta plataforma és molt usada en investigació, que és usada en més de 750 universitats i centres de recerca. És una plataforma en sí, no requereix de programari addicional. És capaç de funcionar en Linux, Windows i MacOS. El llenguatge de programació potser C, C++, Java, Python, Matlab o URBI.

Un altre punt fort és el fet de ser desenvolupat per una empresa, la qual cosa aporta fiabilitat en el producte i garanties de solvència. Per contra, requereix un aprenentatge i adaptació a la plataforma. Es tracta d'una interfície d'usuari amb representació gràfica. La llicència educativa més econòmica té un cost de 320 francs suïssos (246 €).



Imatge 3: Interficie de Webots

4.2.3. Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio és una extra creada per l'empresa Microsoft [7] , integrable a la plataforma Visual Studio per a desenvolupament. La plataforma més els extres afegits permet crear aplicacions de robots. El conjunt resultant és una potent eina de disseny i simulació 3D de robots.

Aquest afegit a plataforma requereix una instal·lació prèvia de Microsoft Visual Studio o Microsoft Visual C#, amb com a mínim la versió .NET 3.5 SP1. També requereix l'execució de DirectX versió 9 o superior. Els programes Microsoft Visual Studio o Microsoft Visual C# requereixen la instal·lació de Microsoft Windows. Per al funcionament dels programes creats, és possible executar en els sistemes operatius Microsoft Windows XP SP2, Microsoft Windows Vista, Microsoft Windows 7, Microsoft Windows Server 2003 R3 i Microsoft Windows Server 2008. La llicència més econòmica de sistema operatiu és Windows 7 Home Premium 149,99€. La llicència més econòmica de plataforma Visual Studio és Visual Studio 2010 Professional 800€. La llicència Microsoft Robotics és gratuïta.

L'eina MS Robotics és molt potent per a la simulació 3D, però aquesta característica no és adient en simulacions 2D on es requereix un càlcul àgil.

Imatge 4: Simulació amb MS Robotics

Imatge 5: Simulació amb MS Robotics

4.3. *Models de robots programables*

En aquesta secció es fa un estudi de robots on poder executar els controladors resultants del projecte. Es fa un estudi de models de robots programables a la venda en aquest moment amb iguals característiques, o bé amb possibilitat de tenir iguals característiques que el robot modelat a la simulació.

Les característiques requerides són: mida similar al robot de neteja, mateixos o més sensors, si fos possible capacitat real d'escombrat.

4.3.1. Roomba de iRobot

Aquest robot netejador és la marca pionera de robots netejadors. És famós a nivell científic per la possibilitat de hackejar-lo, ja que disposa de molta literatura explicativa, així com llibreries i tutorials com [9] .

Fabricant: iRobot [8]

Prestacions: Tots els sensors i moviments requerits i capacitat d'escombrat.

Preu: entre 199,99$ i 599,99$ segons el model

4.3.2. iRobot Create Programmable Robot

Aquest robot té els mateixos que el Roomba però sense aspiradora. És de la mateixa casa que la marca que Roomba. És una versió pensada per a aquells usuaris que compraven un Roomba per a usar-lo per a programar-lo en comptes de usar-lo d'aspiradora.

És un robot específic per a ser programat i disposa de molta documentació detallada en diverses plataformes per a facilitar el seu ús.

Fabricant: iRobot, web del model Create [10]

Prestacions: Tots els sensors i moviments requerits

Preu: 129,99$

Imatge 6: iRobot Create

5. Simulador

5.1. Selecció del simulador

La selecció del simulador consisteix en la tria del simulador més convenient donades les especificacions d'ús requerides.

Per a seleccionar el simulador més convenient, cal plantejar els requeriments:

El simulador ha de permetre un entorn capaç de simular en 2D un terra amb obstacles i un robot circumferència. El simulador ha de permetre definir i emmagatzemar els entorns del robot. El simulador ha de permetre desar en formats externs els resultats de les simulacions. El simulador ha de permetre executar en el robot els controladors dels tipus definits en els objectius. El simulador ha de permetre executar simulacions massives en breus períodes de temps. El simulador ha de permetre un alt grau de flexibilitat en canvis de funcionament i paràmetres interns. Caldrà escollir, en la mesura del possible, una opció que permeti reproduir els resultats del projecte en el màxim de sistemes informàtics. Caldrà escollir, en la mesura del possible, una opció de preu reduït.

En l'apartat Simuladors disponibles al mercat (4.2.) s'ha realitzat un estudi sobre els principals simuladors del mercat i s'han comentat els pros i contres de cada opció.

Després de sospesar les opcions disponibles i comparar-les amb els requisits, s'opta per la creació d'un simulador a mida.

5.2. Disseny i construcció del simulador

Partint dels requeriments exposats en l'apartat anterior, es fa un estudi dels usuaris i casos d'ús.

5.2.1. Estudi dels usuaris i casos d'ús

Realitzat un estudi dels usuaris i els casos d'ús, s'obté el diagrama de classes resultant de la imatge 7.

En la imatge 7 es pot veure com existeix una classe IA que té un objecte de classe Robot. L'objecte Robot que té un objecte de la classe Simulador, un objecte AdministradorSensor i un objecte AdministradorAccio.

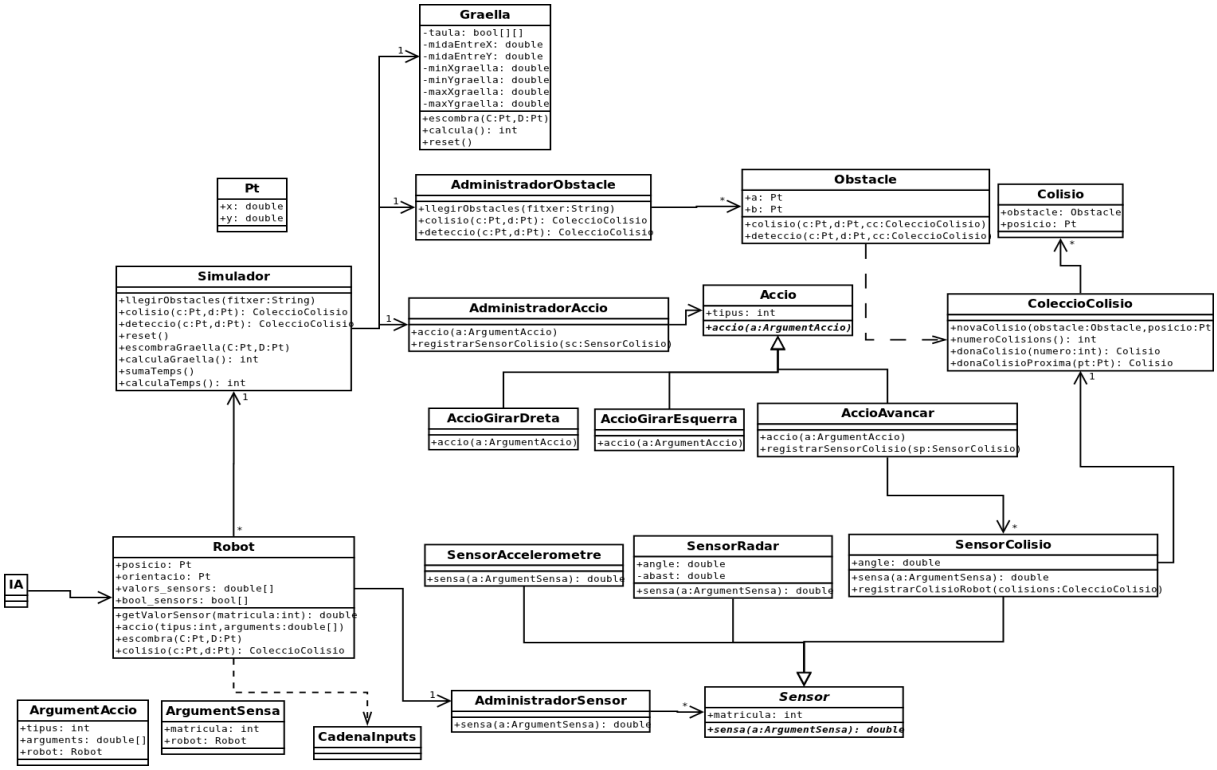
En els propers subapartats s'explica breument les classes més bàsiques que componen el disseny.

5.2.1.1. Classe Robot, classe Simulador i classe IA

La classe Robot és una representació del robot físic. Aquesta classe conté les característiques del robot (mida, tipus de sensors..) i conté la seva representació en el simulador (posició i orientació). La classe robot posseeix un simulador que li calcula l'entorn.

La classe Simulador és la classe que engloba, desa i calcula l'entorn que envolta els robots que posseeix/en al simulador. La classe Simulador conté un AdministradorObstacle (que gestiona tots els segments del pla que el robot no pot travessar), un AdministradorAccio (que gestiona la realització de totes les accions del robot) i un Graella (que gestiona el comptador dels punts escombrats).

La classe IA és el controlador del Robot, és a dir, la part que pren les decisions d'accions i les ordena executar.



Imatge 7: Diagrama de classes

5.2.1.2. Classe Obstacle i AdministradorObstacle

La classe Obstacle és la classe que representa a un segment del pla que el robot no pot travessar. Tal i com es pot veure a la imatge 7, cada objecte Obstacle està definit per dos punts. Obstacle implementa les funcions necessàries per al càlcul de la col·lisió i la detecció. Usa la classe ColeccioColisio explicada en l'apartat següent.

AdministradorObstacle és la classe que agrupa tots els objectes Obstacle. És una classe que serveix per a fer de pont entre el Simulador i tots els Obstacles. El Simulador té un sol AdministradorObstacle i desconeix la classe Obstacle i si n'hi ha algun. AdministradorObstacle implementa la funció de lectura del fitxer i creació de tots els Obstacles.

5.2.1.3. Classe Colisio i classe ColeccioColisio

La classe Colisio és la classe que guarda la relació entre una posició de topada del robot i Obstacle amb el qual ha topat.

La classe ColisioColisions és la classe que agrupa els objectes Colisio. D'aquesta manera queda integrada dins de ColisioColisions totes les gestions de les Colisions, de tal manera que les altres classes només saben de l'existència de ColisioColisions, i només en cas de que la usin (o la passin).

El tractament de les col·lisions es fa a través de l'objecte ColeccioColisio, el qual aglutina objectes colisions. Aquest objecte pertany al SensorColisio i és modificat per l'AccioAvancar.

5.2.1.4. Classe Graella

La classe Graella és una classe que forma part de la classe Simulador i que té com a únic propòsit agrupar els mecanismes de comptabilitzar el pas del robot per l'espai.

La classe Graella es composa principalment d'una matriu de booleans que representen una quadrícula física de punts separats 10 unitats els uns dels altres. L'espai per on circula el robot queda escombrat (els components representatius d'aquella posició passen a valer false).

La classe Graella aglutina les funcions de comptar i actualitzar la matriu de booleans de manera que el conjunt queda en mòdul i fa més fàcil l'ús del Simulador.

5.2.1.5. Classe ArgumentSensa i classes AdministradorSensor i les classes heretades de Sensor

La classe Sensor és una classe abstracta de la qual hereten els diferents sensors que el robot pot tenir. En les classes heretades de Sensor s'implementa la funció sensa(a:ArgumentSensa) per la qual es calcula i retorna el valor del sensor.

La classe ArgumentSensa és la classe que s'usa per a passar els paràmetres de la lectura d'un sensor que es vulgui mesurar. Es composa del nombre del sensor (que es vol llegir) i el robot del qual forma part.

La classe AdministradorSensor és la classe que gestiona tots els sensors. El tractament dels sensors d'un robot i les seves lectures es fa a través de l'objecte AdministradorSensor que aglutina els sensors de que disposa el robot. Aquesta classe fa de pont entre el robot i els seus sensors, de manera que el robot només interactua amb la classe AdministradorSensor i desconeix els objectes Sensor.

La classe SensorAccelerometre serveix per a emular un sensor del robot que sigui capaç de retornar diferents derivades de la posició del robot i de la seva orientació. Per a poder calcular-les, aquesta classe usa uns històrics de posició i orientació que es desen en la classe Robot.

La classe SensorRadar serveix per a emular un sensor de radar del robot amb abast màxim definit. Aquesta classe s'encarrega de retornar un valor entre 0 i 1 que emula la lectura del retorn de l'infraroig en l'angle respecte a l'orientació definit.

La classe SensorContacte serveix per a emular un sensor de contacte. Aquesta classe defineix una obertura de detecció d'angle especificat des de 0 (part frontal del robot) fins a l'angle especificat. El retorn és un valor 0 o 1 segons si hi ha hagut detecció per contacte.

5.2.1.6. Classe ArgumentAccio i classes AdministradorAccio i Accio

La classe Accio és una classe abstracta de la qual hereten les diferents classes d'accions possibles. Les accions hereves implementades són tres : AccioGirarDreta, AccioGirarEsquerra i AccioAvancar. En les classes heretades d'Accio s'implementa la funció accio(a:ArgumentAccio) en la qual es calcula i escriu el resultat de l'acció. En el cas especial de la classe AccioAvancar, a més es calcula la ColeccioColisio que usaran els SensorColisio.

ArgumentAccio és la classe que s'usa per a passar els paràmetres de una acció que es vulgui realitzar. Es composa de el tipus d'acció que es vol realitzar (sigui AccioGirarDreta,AccioGirarEsquerra o AccioAvancar), l'argument (la quantitat a avançar o a girar) i el robot.

La classe AdministradorAccio és la classe que gestiona tots els sensors. Aquesta classe fa de pont entre el simulador i les seves accions possibles, de manera que el simulador només interactua amb la classe AdministradorAccio i desconeix els objectes Accio.

La classe AccioGirarDreta serveix per a moure la orientació del robot un angle determinat cap a la dreta.

La classe AccioGirarEsquerra serveix per a moure la orientació del robot un angle determinat cap a l'esquerra.

La classe AccioAvancar serveix per a intentar moure la posició del robot un espai determinat, gestionar les topades (a través de les classes Obstacle, Colisio i derivades) i iniciar els mecanismes del robot que gestionen les percepcions de les topades.

5.2.2. Entorn i llenguatge de programació

El resultat de l'estudi és implementat en un entorn Mono (.Net) en el llenguatge C# .

Les raons d'aquesta elecció són dues principals: per una banda, C# és un llenguatge en gran increment d'ús, per altra banda la solució s'implementa en un entorn multiplataforma i lliure que permetrà la reproducció dels resultats sense entrebancs.

Donat que el simulador és una eina del projecte, no requereix un entorn de finestres. Per aquesta raó en aquest projecte el simulador s'implementarà en una solució de consola simple.

5.2.3. Sistema d'entrada d'instruccions i paràmetres

Una de les característiques que faran més útil el simulador és com agilitzar l'entrada de paràmetres del robot i de la simulació.

L'entrada dels paràmetres del robot és dual. En les fases inicials del projecte s'usava un fitxer de text pla on eren descrites les característiques del robot que es volia simular. Actualment, a més d'aquesta opció de parametrització des de fitxer, es pot indicar al programa que crei els robots a través de la classe CadenaInputs, la qual va ser creada i usada per a la implementació dels controladors que es descriuen més endavant. Per a les proves experimentals s'ha usat la parametrització a través de la classe CadenaInputs.

El sistema d'entrada de paràmetres per a la simulació (l'entorn) és la lectura i interpretació de l'arxiu d'obstacles. Aquest arxiu és un arxiu de text pla (qualsevol extensió) on cada línia és una successió de quatre doubles separats pel caràcter ";" i que representen els dos punts (Ax;Ay;Bx;By) del pla que defineixen un segment, que és la paret física.

Finalment, la definició del tipus de simulació i els paràmetres de la mateixa s'han resolt creant les classes Problema, de tal manera que establint uns pocs paràmetres en la MainClass, un usuari novell seria capaç de fer servir el simulador sense més problemes.

```
        // parametres:
        string parets = "zdades/entrenament.txt";
                  int passossimulacio ;
                  Pt posicio = new Pt(30,30);
                  Pt orientacio = new Pt(1.0,0.0);
                  bool mostraperpantalla;
                  int maximCompetidors ;
                  int nombrecombats;
                  bool combatdedos;
                  bool testeigpoblacioinicial;
                  bool entrenar,executar;
        int iteracions_entrenament = 1000;

                  int numoctogon = 0;
                  int numoctogon2 = 0;
                  int numrandom = 0;
                  int numrandomparametre =0 ;
                  int numparametreparametre =0;
                  int nummatriu =0;
                  int nummatriurandom=0;
                  int numRL =10;
                  // fi parametres
```

Codi 1: Definició de variables necessàries i suficients per a executar les simulacions

Tot i que C# permet la interpretació de strings, i per tant seria permès fer executar un codi entrat des un arxiu (com passa amb els obstacles i el robot), s'ha rebutjat aquesta opció pel gran perill d'introducció d'errors en la fase d'interpretació de l'arxiu llegit.

5.2.4. Sistema de recol·lecció de dades

Una de les parts més importants a l'hora d'usar el simulador és la capacitat de recollir les dades rellevants provinents de l'execució de les simulacions.

Per a permetre el posterior tractament d'aquestes dades, s'ha dotat el simulador de funcions de gravat de les dades resultants d'una simulació en un arxiu de text en format separat pel caràcter ";". A partir d'aquestes dades, s'han creat uns scripts que permeten aglutinar tots els resultats de la carpeta en un únic fitxer de text, també en format separat per ";", que és trivialment importat a l'excel.

A més, s'ha creat una funció generadora del log de la simulació per al bolcat de les dades resultants de les tandes de simulacions i tornejos (en el cas d'algorismes genètics) així com les característiques de cadascun, els paràmetres i puntuacions de poblacions inicials i finals i la data de l'execució.

Per raons de documentació de proves, també s'ha dotat al simulador de funcions de desat i importació dels paràmetres que conformen el controlador, és a dir, per a permetre el desat i recuperació dels controladors entrenats/evolucionats .

Com a complement de la informació numèrica, es dóna la possibilitat de mostrar per pantalla en mode visual (amb caràcters) l'execució de la simulació alentida en 50ms per pas (per a poder captar-la).

6. Banc de proves

El banc de proves és el mètode que s'usa per a testar i mesurar l'eficàcia dels controladors resultants. En aquesta secció s'explica els bancs de proves usats per a l'entrenament i avaluació dels controladors.

Els bancs de proves creats per al simulador (o a l'inrevés) consisteixen en definir un espai delimitat per Obstacles (rectes que el robot no pot traspassar). Sobre aquest espai s'escampen els punts de la Graella (que com s'ha dit al capítol *Classe Graella* 5.2.1.4. els punts estan separats per 10 unitats) de tal manera que el robot es mou limitat per l'entorn mentre amb la graella es quantifica la superfície abastada mitjançant l'escombrat de punts.

S'ha pensat aquest mètode perquè és un mètode efectiu i lleuger per a mesurar la superfície recorreguda pel robot.

La manera com es guarden els entorns es mitjançant un fitxer de text. A cada línia del fitxer de text trobem un Obstacle representat per quatre valors double separats pel caràcter ";". Amb aquesta representació dels obstacles es construeixen els objectes Obstacle de l'AdministradorObstacle.

El nom del banc de proves a usar s'indica en el MainClass en la variable parets.

S'han usat tres entorns, anomenats *"entrenament.txt"*, *"menjador_petit.txt"* i *"menjador_mitja.txt"*.

6.1.1. "Entrenament.txt"

El principal banc de proves per als algorismes usat ha sigut l'arxiu "entrenament.txt"

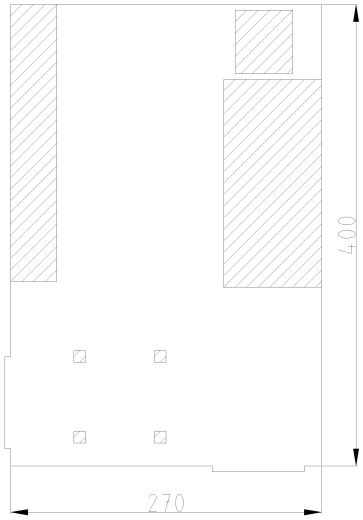
```
# quadrat parets
0;0;0;300
300;0;300;300
0;300;300;300
0;0;300;0
# sofa enmig
80;120;210;120
210;120;210;180
80;180;210;180
80;120;80;180
```

Codi 2. Arxiu "entrenament.txt"

Com es pot veure en el codi 2, es tracta d'un entorn molt senzill només vuit segments. La seva superfície màxima és 82200 unitats i els punts màxims a escombrar són 743.

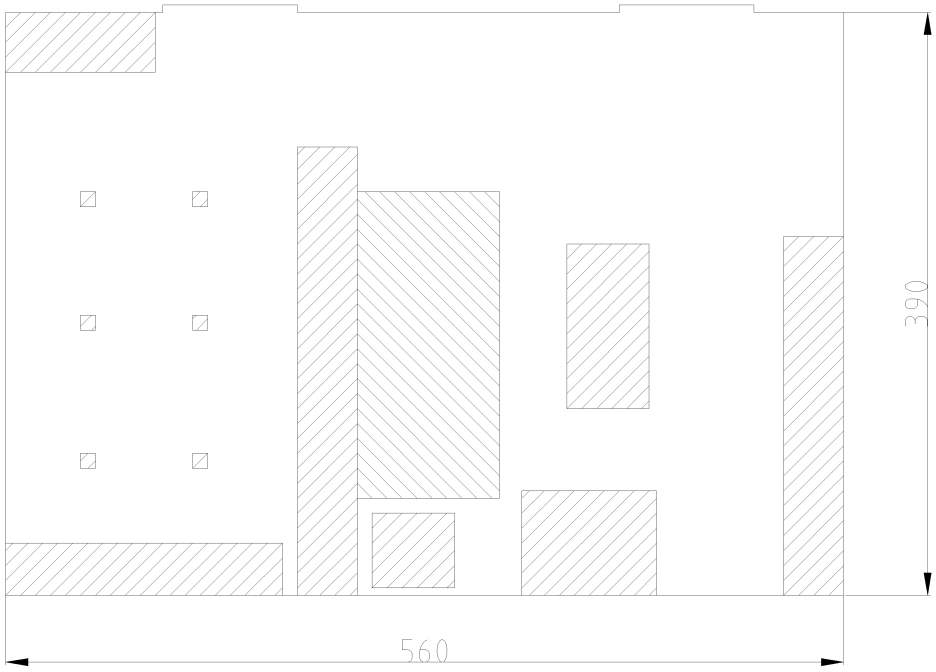
6.1.2. "Menjador_petit.txt"

Aquest banc de proves simula un menjador de mides reals (les unitats es prenen com a centímetres). La seva superfície màxima és de 78400 unitats quadrades i els punts màxims són 7614.



6.1.3. "Menjador_mitja.txt"

Aquest banc de proves simula un menjador de mides reals (les unitats es prenen com a centímetres). La seva superfície màxima és de 156725 unitats quadrades i els punts màxims són 15496.



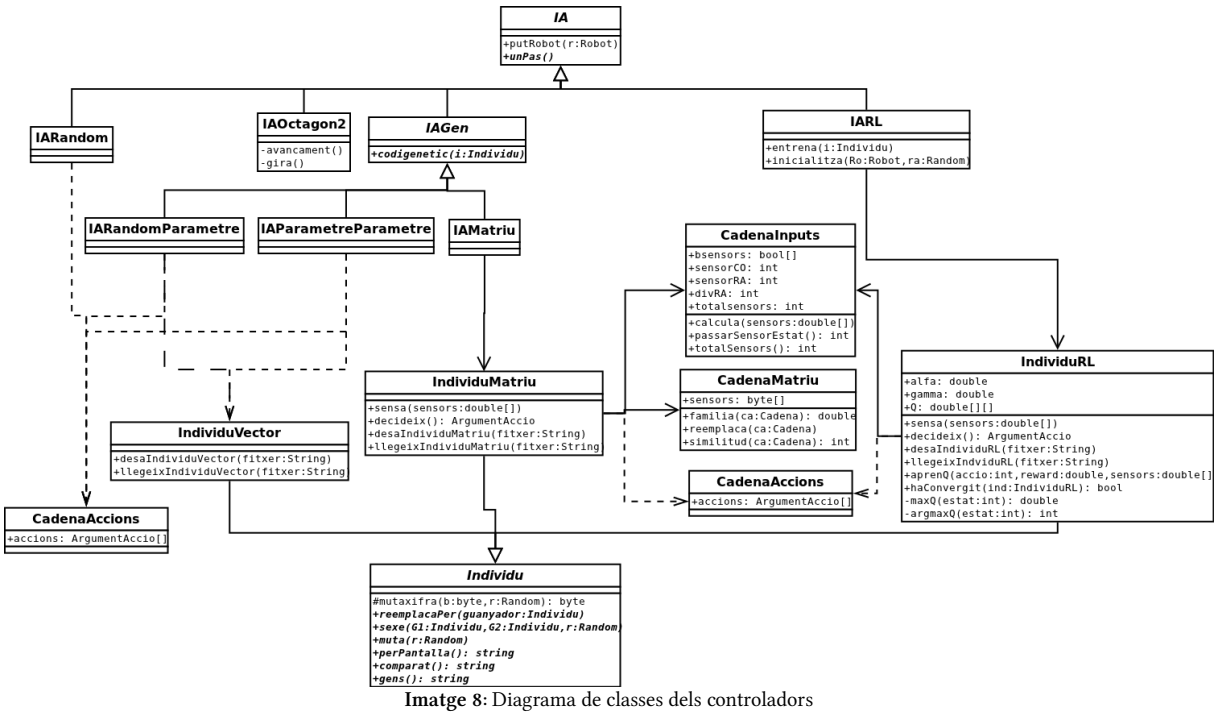
7. Controladors creats

En aquest apartat s'explica els diferents controladors creats basats en algorismes, el seu disseny, les seves possibilitats i les seves diferències.

Tal i com es va veure explicar en el capítol *Estudi dels usuaris i casos d'ús* (5.2.1.), el controlador dins en diagrama de classes del sistema (Imatge 7) és un objecte de classe IA que posseeix un objecte Robot i aquest alhora, posseeix altres objectes, entre ells un objecte Simulador.

IA és en realitat una classe abstracta de la qual hereten totes les classes de controladors. Totes elles comparteixen la funció unPas(), que és la funció més important de les IA: primer decideix una acció a fer i després l'executa.

S'ha creat un total de sis controladors: IARandom i IAOctagon2 (replicades per a proves de control), IARandomParametre, IAParametreParametre i IAMatriu (basades en computació evolutiva) i IARL (basada en aprenentatge per reforç).



Imatge 8: Diagrama de classes dels controladors

En la imatge 8 es pot veure com totes les classes no abstractes que hereten de la classe IA (excepte IAOctogon2) es serveixen de la classe CadenaAccions per a definir els arguments de les seves accions. CadenaAccions és una classe estàtica que només serveix per a contenir en un únic lloc les accions (els objectes ArgumentAccio) comuns a les classes heretants de IA que requereixin aquesta unificació. CadenaAccions conté també l'acció “estarquiet” la qual és una acció (un objecte ArgumentAccio) els parametres del qual són avançar 0 unitats.

D'igual manera, tant IAMatriu com IARL a través dels seus objectes (IndividuMatriu i IndividuRL respectivament) posseeixen un objecte CadenaInputs que defineix el robot mitjançant els sensors del robot i

els rangs de divisió per a cada sensor. El robot de la simulació també obté la definició del robot a través de l'objecte CadenaInputs per a prevenir incongruències i unificar les especificacions.

7.1. Classes comuns per a estandarització de comparacions

En aquest apartat es fa un resum de les classes CadenaInputs, CadenaAccions i les classes Individu per tal d'establir les bases comunes a tots els controladors basats en algorismes.

7.1.1. CadenaInputs

La classe CadenaInputs serveix alhora per a definir les característiques del robot i per a gestionar la interpretació de les lectures dels sensors del robot.

Aquesta classe és important perquè estandaritza aquelles classes que llegeixen dels sensors.

```
public bool[] indicadors;

public static int sensorAC = 4;
public static int sensorRA = 3;
public static int sensorCO = 2;
public static int sensorCOvirtual = 1;
public static int divRA = 3;
public static int histMaxim = 4;
public static int totalsensors = sensorAC  +sensorRA + sensorCO;
private static int totalindicadorsboleans = sensorAC  +sensorRA*divRA + sensorCO + sensorCOvirtual;
public static int max_estats = (int)Math.Pow(2,sensorAC)*(int)Math.Pow(Math.Pow(2,auxestatsdivRA),sensorRA)
                          *(int)Math.Pow(2,sensorCO);
public static double[] colocacioradar = new double[5]{0,Math.PI/4,-Math.PI/4,Math.PI/2,-Math.PI/2} ;
public static double[] colocaciocolisio = new double[2]{Math.PI/2,-Math.PI/2} ;
public static double[] colocaciocolisiovirtual = new double[1]{0.0} ;
public static int[] colocaaccelerometre = new int[8]{1,-1,2,-2,3,-3,0,-histMaxim};
// el num es la derivada i ha de ser menor a histMaxim per a poderla calcular
// a causa de que c# no distingeix entre 0 i -0, el -0 es canvia per un -histmaxim
private static int auxestatsdivRA = 2; // si es toca divRA, actualitzar aquest

public static double abastradar = 100.0;
public static double radidelrobot = 20.0;
```

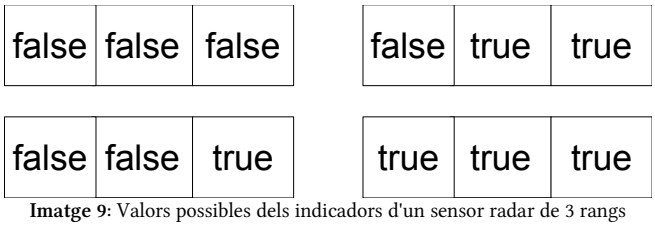
Codi 3: Definició del nombre de sensors i rangs de sensors.

La classe CadenaInputs conté un vector de booleans (anomenat indicadors) que representen cada booleà un indicador de sensor o de rang de sensor. Aquest vector és tant llarg com totalindicadorsbooleans.

És important fer notar que el vector indicadors correspon a la interpretació de les dades recollides en el vector de doubles que prové dels sensors. És a dir, CadenaInput processa les lectures dels sensors i les transforma en informació útil. La transformació de lectures a informació útil és diferent per a cada tipus de sensor.

El valor de l'indicador en el cas dels sensors accelerometres, serà true si la lectura és positiva i l'indicador serà false si la lectura és negativa o zero.

El valor dels indicadors de cada sensor radar serà en funció del valor de la lectura del radar. Per a cada sensor radar, poden haver-hi diverses divisions, reflexades en el valor divRA de CadenaInput. La definició de si l'indicador és true o false, és la següent: Aquell indicador en el rang del qual es trobi el valor de la lectura serà true. Els indicadors anteriors (fins al robot) seran false, i els posteriors (fins al final de l'abast del sensor) seran true.



Imatge 9: Valors possibles dels indicadors d'un sensor radar de 3 rangs

El valor dels indicadors de col·lisió reals serà true si el valor del sensor és 1, i serà false si el valor del sensor és 0. Els valors dels indicadors de col·lisió virtuals, serà true si els sensors de col·lisió que el rodegen són tots dos 1, i serà false en cas de que no sigui així.

Les definicions de les conversions de valor de sensor a indicadors, les definicions de quants indicadors existeixen i el mecanisme per al seu ús només es troben a la classe CadenaInputs i per tant aquesta classe formarà part d'aquelles classes que faran ús dels sensors.

Els indicadors triats han sigut:

- Quatre sensors d'acceleració, corresponents a la velocitat lineal, la velocitat angular, l'acceleració lineal i l'acceleració angular.

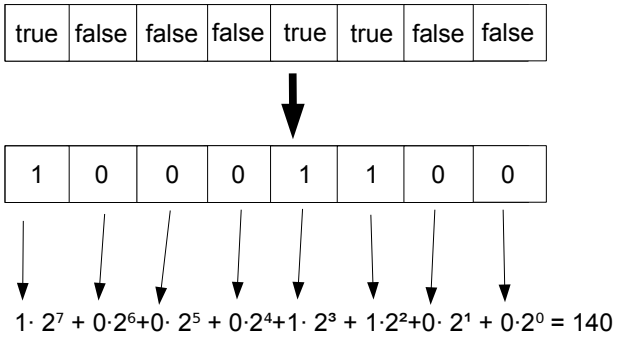
- Nou indicadors de radar, corresponent als sensors de radar situats a 0 radians respecte l'eix d'orientació del robot, $\pi/4$ radians respecte l'eix d'orientació i $-\pi/4$ radians respecte l'eix d'orientació. Cada sensor ha sigut dividit en 3 indicadors equitatius, per a detecció d'obstacles de més aprop a més lluny.

- Dos sensors de col·lisió reals, corresponents a l'angle entre 0 radians i $\pi/2$ radians respecte l'eix d'orientació i l'angle 0 radians i $-\pi/4$ radians respecte l'eix d'orientació

- Un sensor de col·lisió virtual corresponent a un sensor virtual en la posició 0 radians respecte l'eix de la orientació.

La classe CadenaInputs realitza una conversió prou important per a ser mencionada en la memòria, es tracta de la conversió del vector d'indicadors booleans a un enter estat. Aquesta conversió es troba en la funció passarIndicadorsaEstats().

La funció passarIndicadorsaEstats() converteix un vector indicador a un sol enter estat, evitant estats impossibles per a un mateix sensor. Aquesta funció està inspirada en la conversió de números escrits en bits a enters (prenent els valors true com a 1 i false com a 0).



Imatge 10: Conversió inicial de bits a enter

En un inici, la transformació de vector d'indicadors a estat enter seguia l'algorisme de la imatge 10, però donat que existeixen configuracions d'indicadors impossibles de donar-se, aquest mètode es va reformar en

l'actual transformació per a impedir la proliferació d'estats impossibles per configuració i/o càlcul d'indicadors.

El punt clau d'aquesta transformació es quedar-se amb la informació rellevant i eliminar les configuracions d'indicadors impossibles per construcció i/o càlcul. Per a fer-la cal distingir cada indicador respecte al tipus i la configuració amb el que s'ha calculat.

```
public int passarIndicadorsaEstat()
          {
          int total = 0;

          //sensors accelerometre
          // cada sensor un estat
          for(int i = 0; i < sensorAC ; i++)
          {
                    total = total*2 + (indicadors[i]?1:0);
          }

          // sensors radar
          // cada radar fisic te divRa indicadors
          for (int j = 0 ;j < sensorRA; j ++) // j es el numero de sensor fisic
          {
                    int k = 0;
                    int i = 0;
                    while (i < divRA && 0==k )// i ens el numero de divisio
                    {
                              if(true == indicadors[sensorAC+ j* divRA +i ] )
                              // uso el true o false de l'indicador
                              {
                                        k = i;
                              }
                              i = i++;
                    }
                    total = total*auxestatsdivRA + k;

          }

          // sensors colisio reals
          for(int i = sensorAC  + sensorRA*divRA ; i < sensorAC +sensorRA*divRA + sensorCO;

          {
                    total = total*2 + (indicadors[i]?1:0);
          }

          // sensors de colisio virtuals no tenen estat pq son implicits dels estats dels

          return total;

}
```

Codi 4: Funció passarIndicadorsaEstat()

Cada sensor acceleròmetre representa per ell mateix una entitat i per tant per a cada valor que prengui (true o false) cada sensor, fa incrementar els sensors totals en en doble. Els sensors acceleròmetres segueixen la conversió inicial de la Imatge 10 com queda reflectit al Codi 4.

En canvi en els sensors radar, donada la codificació dels indicadors de cada sensor radar físic (és a dir, els rangs per cada sensor radar físic), obtenim tota la informació rellevant d'un radar físic sabent el rang (a prop, lluny, més lluny ...) on comencen les lectures true, per tant, només caldrà reflectir a l'estat la posició del primer true. Per tant la representació a l'estat d'un sensor radar depèn dels rangs lògics que CadenaInput imposi per a calcular els indicadors, i aquests rangs estan especificats a la variable estàtica divRA (Codi 3). Per a indicar al sistema per quin número ha de multiplicar el total anterior en el càlcul, es defineix a l'inici de CadenaInput la variable estàtica auxestatsdivRA, la qual simplement indica quants bits serien necessaris per a representar el nombre màxim possible de rang a representar (és a dir quants bits són necessaris per a representar divRA). Donat que auxestatsdivRA és el logaritme de base 2 de divRA, aquesta conversió permet estalviar estats fútils de manera quadràtica, passant dels 2^{divRA} estats inicials segons Imatge 10 per sensor radar als només $2^{auxestatsdivRA} = 2^{\log2(divRA^*)} = divRA^*$ estats.

Els sensors de col·lisió reals són representatius en sí mateixos, així que segueixen la conversió inicial de la Imatge 10 com queda reflectit al Codi 4.
En canvi, els sensors de col·lisió virtuals es calculen en funció de la configuració dels sensors de col·lisió reals, i per tant, no codifiquen cap estat nou.

En conclusió, la conversió del vector d'indicadors booleans a estats dóna un nombre total d'estats de:

$$Total\ estats = 2^{sensorAC} \cdot \left(divRA^{*}\right)^{sensorRA} \cdot 2^{sensorCO} \quad \textbf{(11)}$$

On divRA* és $2^{auxestatsdivRA}$.

7.1.2. CadenaAccions

La classe CadenaAccions és l'objecte estatic on s'emmagatzemen les accions vàlides del sistema.

La classe CadenaAccions estandaritza sis accions bàsiques més una acció sense moviment. Les tres primeres accions són accions de poc moviment, en canvi les tres següents són accions de gran moviment (el doble).

Les tres primeres accions són equivalents a l'esperança matemàtica de multiplicar els arguments de les tres últimes accions per una distribució uniforme.

Usant únicament les accions definides a CadenaAccio és possible escombrar completament el banc de proves Entrenament.

```
public static class CadenaAccions
	{
		private static ArgumentAccio decisio0 = new ArgumentAccio(0,new double[] {Math.PI/4} );
		private static ArgumentAccio decisio1 = new ArgumentAccio(1,new double[] {Math.PI/4} );
		private static ArgumentAccio decisio2 = new ArgumentAccio(2,new double[] {10.0} );
		private static ArgumentAccio decisio3 = new ArgumentAccio(0,new double[] {Math.PI/2} );
		private static ArgumentAccio decisio4 = new ArgumentAccio(1,new double[] {Math.PI/2} );
		private static ArgumentAccio decisio5 = new ArgumentAccio(2,new double[] {20.0} );

		public static ArgumentAccio estarquiet = new ArgumentAccio(2,new double[] {0.0}  );

		public static ArgumentAccio[] accions = new ArgumentAccio[]
		{
			decisio0,
			decisio1,
			decisio2,
			decisio3,
			decisio4,
			decisio5
		} ;

	}
```

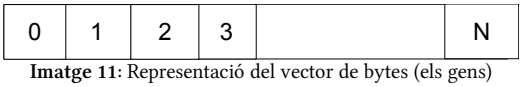
Codi 5: CadenaAccions

7.1.3. Individu, IndividuVector, IndividuMatriu i IndividuRL

La classe individu és una classe abstracta que serveix per definir les funcions estàndard que són implementades en les classes que n'hereten.

En la primera IA realitzada (amb les teories de computació evolutiva) es va veure la conveniència de desar en algun objecte amb sentit per ell mateix i fora de la IA el codi genètic i les funcions que implementen els mètodes usuals. Posteriorment es va afegir la necessitat de desar i recuperar individus des d'arxius. Dins de la classe Individu es troben les funcions de desat i recuperació de la informació que compon un individu (sigui els gens o la Q). També dins d'Individu es troba la funció *mutaxifra* que donat un byte d'entrada i un objecte Random, retorna la xifra entrant amb ±1 en un 60% dels casos, amb un ±10 en un 30% dels casos i retorna una xifra aleatòria en el 10% restant.

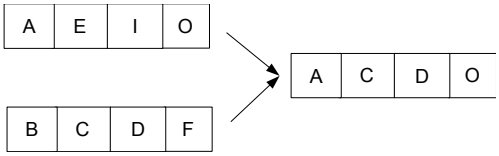
IndividuVector és una classe genètica i la classe més simple derivada de Individu. El codi genètic de IndividuVector es composa d'un vector de bytes de llargada establerta en la funció de creació. IndividuVector té implementades les funcions pròpies de mutació i reproducció.



Imatge 11: Representació del vector de bytes (els gens)

En la funció de mutació (*muta*) es muten tots els gens (usant la funció *mutaxifra*) en un 0,5% dels casos, es muten 3 gens en un 0,5% dels casos, es muten 2 gens en un 4,5% dels casos i es muta només un gen en el 95% dels casos.

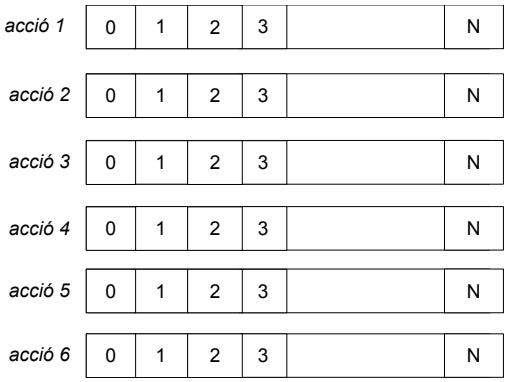
En la funció reproducció (*sexe*) els gens de l'individu són reemplaçats per una barreja dels dos individus entrats com a paràmetres. En un 5% dels casos, els nous gens són els gens mutats del primer individu dels paràmetres,i del 95% restants, un 70% correspon als gens del primer individu amb un gen del segon, en un 25% corresponen als gens del segon individu amb dos gens del segon i en un 5% corresponen als gens del primer amb 3 gens del segon.



Imatge 12: Exemple de reproducció amb 4 gens

Com s'ha dit en el capítol *Sistema de recol·lecció de dades*(5.2.4.), s'ha implementat en IndividuVector uns mecanismes de desat i recuperació del codi genètic que conforma la identitat d'un individu. D'aquesta manera, un individu es pot desar per a ser recuperat en una altra ocasió.

IndividuMatriu és la classe Individu usada per a genètica més similar a la classe usada en aprenentatge per reforç. El codi genètic de IndividuMatriu es composa d'objectes CadenaMatriu, que es composats per un vector de bytes. Hi ha una CadenaMatriu per a cada acció possible. Les Cadenes són tan llargues com indicadors estan definits a CadenaInput (que és l'objecte encarregat de convertir les lectures dels sensors en indicadors). Les accions de IndividuMatriu estan definides a CadenaAccions (Codi 5).



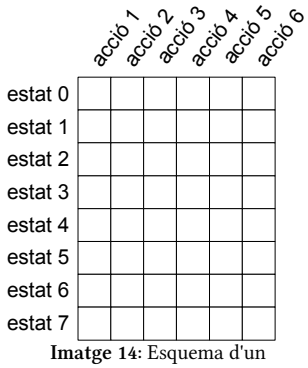
Imatge 13: IndividuMatriu. Cada posició conté un byte

En la funció de mutació (*muta*) es genera un nou individu de manera aleatària i aquest nou individu i l'actual es reprodueixen i el fill reemplaça l'individu actual.

En la funció reproducció (*sexe*) els gens de l'individu són reemplaçats per una barreja dels dos individus entrats com a paràmetres. En cas de que els dos individus pares siguin exactament iguals, l'individu es substitueix per un pare i es fa una mutació (*muta*). En cas de que els pares siguin diferents, en un 1% dels casos es produeix la mutació (*muta*) prenent com a inidividu inicial el primer pare. En cas de que els pares siguin diferents, l'individu es reemplaça pel primer pare i s'entra en un bucle del que es té una probabilitat de sortir del 90% a cada volta. En aquest bucle hi ha un 5% de probabilitats de reemplaçar una cadena sencera amb una cadena del segon pare, un 90% de probabilitats de reemplaçar un sol element d'alguna cadena per l'equivalent del segon pare i un 5% de probabilitats de reemplaçar tots els elements d'un mateix indicador (una sensibilitat a un indicador).

Igual que IndividuVector, s'ha implementat en IndividuMatriu uns mecanismes de desat i recuperació del codi genètic que conforma la identitat d'un individu. D'aquesta manera, un individu es pot desar per a ser recuperat en una altra ocasió.

IndividuRL és la classe usada en aprenentatge per reforç. IndividuRL defineix una matriu Q de dos dimensions que conté doubles. Una dimensió de la matriu la formen les accions possibles, que també venen definides a la classe CadenaAccions. L'altre dimensió correspon als estats del sistema. S'ha definit un estat com el vector dels booleans dels indicadors dels sensors. Per a facilitar la seva implementació, aquest vector de booleans es pren com un vector de bits i s'interpreta com a enter. Mitjançant aquesta transformació (que es dóna dins l'objecte CadenaInputs), cada estat queda representat per un nombre, sent els estats totals definits a l'equació 11. IndividuRL té dos funcions destacades: aprenQ i decideix.



Imatge 14: Esquema d'un IndividuRL. Cada posició conté un double

En la funció aprenQ, IndividuRL modifica els valors de la seva matriu Q a través de l' equació 10 descrita en el capítol d'*Aprenentatge per diferencia temporal* (4.1.2.2.). L'aprenentatge es dóna exclusivament a través d'aquesta funció. Els paràmetres d'aprenentatge gamma i alfa són constants en cada execució.

En la funció decideix, IndividuRL fa servir la seva matriu Q (ja apresa) per a escollir una de les accions possibles a partir de l'estat en que es troba (previa lectura dels sensors i trasllat a estat) ja que com hem vist en l'apartat *Problemes de decisió de Markov* (4.1.2.1.), l'acció òptima per a un estat, és l'acció amb valor (de Q[estat][accio]) més elevat.

Igual que en IndividuVector i IndividuMatriu, s'ha implementat en IndividuRL uns mecanismes de desat i recuperació del codi genètic que conforma la identitat d'un individu. D'aquesta manera, un individu es pot desar per a ser recuperat en una altra ocasió.

7.2. Controlador amb l'algorisme del robot ASTAN BL001A: IAOctagon2

Aquest projecte es va iniciar com a mètode per a millorar l'eficàcia del robot ASTAN BL001A, així que sembla convenient poder comparar l'eficàcia dels controladors resultants amb el controlador del robot.

Amb l'objectiu de poder comparar els resultats obtinguts pels sistemes de navegació finals amb uns resultats de referència, s'ha cregut necessària la replicació de l'algorisme del robot original.

L'algorisme original consisteix en un recorregut octogonal en espiral creixent fins que el robot detecta una paret. En el moment que detecta la paret, la segueix i a partir de les mesures de la paret, segueix un recorregut mentre esquiva obstacles. Aquesta informació sobre l'algorisme del robot està explicada en el manual d'instruccions del mateix.

S'ha creat una classe IAOctagon2 que implementa l'algorisme del robot original segons les instruccions disponibles en el manual d'usuari.

7.3. Controlador amb comportament aleatori: IARandom

D'igual manera que en l'apartat anterior, per tal de situar els propers resultats en context, s'ha cregut convenient crear un controlador amb comportament aleatori. Aquest controlador pot executar tres accions amb una probabilitat d'un terç cadascuna.

Les accions que pot executar són les primeres tres accions de l'objecte abstracte CadenaAccions, que corresponen a les accions de girar a dreta $\pi/4$ radians, girar a esquerra $\pi/4$ radians i avançar 10 unitats. S'ha escollit aquestes accions perque són equivalents a l'esperança matemàtica de multiplicar les tres últimes accions per una distribució uniforme, distribució que es seguirà en següents controladors.

7.4. Controlador per computació evolutiva

En aquest apartat es comenten les tres implementacions de controladors creats a partir de les teories dels algorismes genètics.

Els controladors creats per computació evolutiva s'implementen en classes derivades de la classe abstracta IAGen, la qual deriva de IA. Aquests controladors fan servir les classes IndividuVector i IndividuMatriu que han sigut explicades a l'apartat *Individu, IndividuVector, IndividuMatriu i IndividuRL* (7.1.3.). Les teories en que es basen han sigut explicades a l'apartat *Algorismes genètics* (4.1.1.1.).

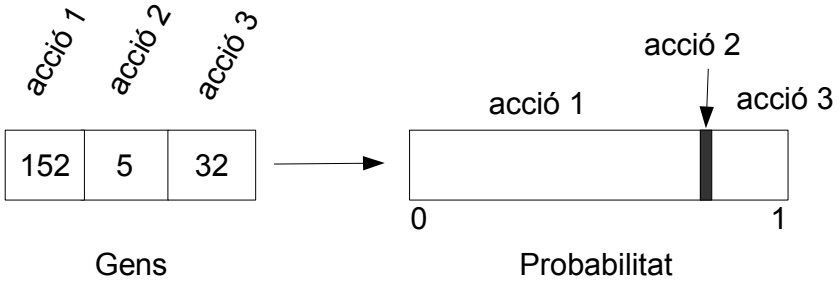
L'avaluació de l'individu consisteix en determinar la puntuació obtinguda per un individu en l'escombrat sota unes condicions d'entorn (els obstacles), un punt d'inici i orientació, un nombre determinat de cicles (anomenats passos de la simulació). A cada pas, el robot fa una acció, i aquesta acció pot generar punts escombrats o no. El recompte de punts escombrats i inicialitzat el realitza el simulador (en qualsevol moment en què es requereixin).

S'han seguit dos mètodes per a la selecció de la següent generació, anomenats Competició de 2 i Competició de 4. En la competició de 2, dos individus són avaluats en les mateixes condicions, i el perdedor és reemplaçat pel guanyador mutat. En la competició de 4, es fan dos parelles i per a cada parella, s'avaluen els membres i es determina un guanyador i un perdedor. Els dos perdedors són reemplaçats per fills dels dos guanyadors.

7.4.1. Comportament aleatori amb probabilitat fixada per gens i paràmetres de moviment fixats: IARandomParametre

En aquest apartat es descriu el controlador creat per algorismes genètics en la classe IARandomParametre.

L'algorisme de la classe IARandomParametre es basa en un IndividuVector de 3 gens. Aquest controlador només pot decidir entre tres accions, que són les primeres accions definides a CadenaAccio (girar a dreta $\pi/4$ radians, girar a esquerra $\pi/4$ radians i avançar 10 unitats). Cada gen s'associa a una acció de tal manera que el valor del gen associat entre la suma dels valors de tots els gens, és la probabilitat associada a aquella acció.



Imatge 15: Interpretació dels gens

Les probabilitats de cada acció s'ordenen en el mateix ordre que les accions. La manera de triar una acció nova és treure un nombre aleatori entre 0 i 1 i mirar en quina franja ha caigut. L'acció a prendre és la que correspon a la franja on ha caigut la variable aleatòria.

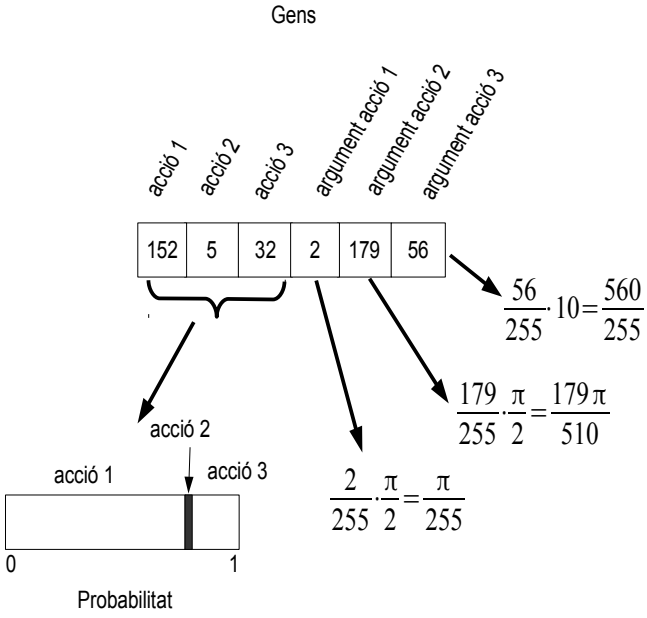
El mètode per a obtenir la següent generació ha sigut la competició de 2 o la competició de 4, ja que s'han realitzat proves d'ambdós mètodes (l'ús d'un o l'altre és indicat a la gràfica dels resultats).

7.4.2. Comportament aleatori amb probabilitat fixada per gens i paràmetres de moviment per gens: IAParametreParametre

En aquest apartat es descriu el controlador creat per algorismes genètics en la classe IAParametereParametre.

L'algorisme de la classe IAParametreParametre es basa en un IndividuVector de 6 gens. Aquest controlador decideix entre tres accions, les quals venen determinades pels seus gens. Els primers tres gens s'usen per a determinar la probabilitat de cada acció, exactament igual que a la classe IARandomParametre.

Els tres últims gens, determinen quines són les tres úniques accions que aquest individu en concret pot realitzar. El càlcul de les accions a realitzar usa les tres últimes accions definides a CadenaAccio (girar a dreta $\pi/2$ radians, girar a esquerra $\pi/2$ radians i avançar 20 unitats). El gen és un byte i per tant pren valors des de 0 fins 255. El càlcul de les accions possibles per a cada individu modifica l'argument de cada acció multiplicant l'argument original pel quocient del valor del gen entre 255. Així, l'espai d'accions possibles en tota la població és finit i concretament hi ha 768 accions possibles.



Imatge 16: Interpretació dels gens

Donat que en la generació de la població inicial (el conjunt dels individus) els gens es creen aleatòriament amb una distribució uniforme, és assumible que el valor mig total de l'argument per a cada tipus d'acció és l'esperança matemàtica de multiplicar els argument de les tres últimes accions per una distribució uniforme i per tant, és igual al valor de l'argument de les tres primeres accions definides a CadenaAccio.

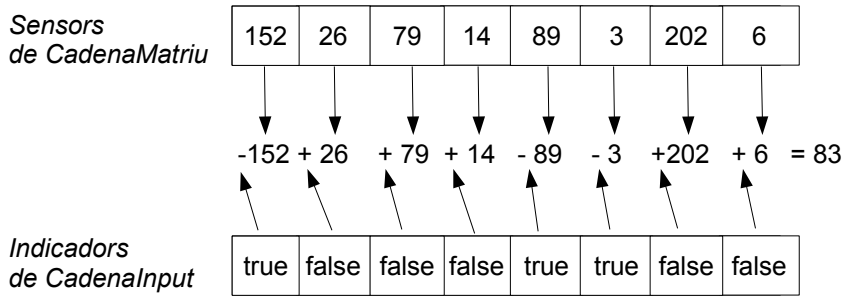
El mètode per a obtenir la següent generació ha sigut la competició de 4.

7.4.3. Comportament calculat segons els gens i els sensors: IAMatriu

En aquest apartat es descriu el controlador creat per algorismes genètics en la classe IAMatriu.

L'algorisme de la classe IAMatriu es basa en un IndividuMatriu de 6 accions i el total dels indicadors de CadenaInputs (totalindicadorsbooleans). Aquest controlador decideix entre sis accions, les quals venen determinades per CadenaAccions (usant totes les accions).

Els gens de l'IndividuMatriu s'articulen en CadenaMatrius. Cada CadenaMatriu és compon d'un vector de bytes de llargada totalindicadorsbooleans (CadenaInputs). A cada decisió, es pren el valor dels sensors del robot i aquestes lectures són introduïdes a l'objecte CadenaInput de l'IndividuMatriu. Dins de CadenaInput es transformen les lectures en indicadors booleans usables agrupats en un vector contingut en l'objecte CadenaInput. Per a determinar quina serà la nova acció, s'usa aquest vector d'indicadors. Per a cada possible acció, hi ha una CadenaMatriu. Dins aquesta CadenaMatriu hi ha la funció resultat, que requereix com a paràmetre un vector de booleans de la mateixa mida que CadenaMatriu. La funció resultat calcula un resultant prenent cada posició del vector de bytes de CadenaMatriu com un pes multiplicat per -1 o 1 (segons si el booleà de la mateixa posició del vector de booleans introduït és a true o false). La suma de tots els pesos multiplicats per 1 o -1 segons correspongui conformen el valor de l'acció. És calcula el valor de totes les accions en aquell moment i es realitza l'acció amb més valor.



Imatge 17: Càlcul del valor d'una acció per a una Cadena de llargada 8

El mètode per a obtenir la següent generació ha sigut la competició de 4.

7.5. Controlador per Reinforcement Learning

En aquest apartat es descriu el controlador creat per aprenentatge per reforç en la classe IARL.

L'algorisme de la classe IARL es basa en un IndividuRL amb una Q de 6 accions. El nombre d'estats màxims es pot trobar a CadenaInput a la variable max_estats. Aquesta IA decideix entre sis accions, les quals venen determinades per CadenaAccions (usant totes les accions). Com s'ha vist en el capítol *Individu, IndividuVector, IndividuMatriu i IndividuRL* (7.1.3.), la Q és una matriu de doubles.

Agafant tots els indicadors disponibles com a estats per a la configuració de CadenaInputs descrita al Codi 3 i seguint l'equació 11, surt un espai d'estats de $2^4 \cdot 4^3 \cdot 2^2 = 4096$ estats. Aquest nombre d'estats tant enorme i que implica un entrenament de milions d'unitats temporals, s'ha comprovat experimentalment que pot ser reduït sense afectació significativa (a diferència del genètic) si es desprecien els sensors d'acceleració en el còmput de l'estat. Mitjançant aquesta simplificació, l'espai d'estats es redueix a 256 estats possibles amb la qual cosa es pot entrenar el conrolador en un temps raonable (passant de milions a uns milers) a més d'aconseguir simplificar la complexitat d'un sistema dissenyat per a només sis accions.

7.5.1. Entrenament clàssic

El procés d'aprenentatge és la modificació de la matriu Q a través de l'equació 10. En el procés clàssic de creació d'un controlador per aprenentatge per reforç s'observen dos etapes: primer cal que Q aprengui i

després mitjançant la Q, el controlador ja pot decidir escollint per a cada estat, l'acció amb valor Q[estat] [accio] més alt.

En un primer moment s'ha intentat un entrenament de Q mitjançant el mètode clàssic. En les diverses proves fetes s'han variat l'alfa, la gamma, s'han fet variables durant el propi entrenament.

Els entrenament s'han fet amb entorns seguint un moviment aleatori primer i després usant la Q per a modificar l'aleatorietat. Posteriorment es van incloure canvis de posició del robot a l'entorn aleatoris i reembrutament de l'entorn.

S'han usat dos funcions de recompensa. Primer es prenia la recompensa com els punts escombrats per l'acció que provoca el canvi d'estat, és a dir, la diferència entre els punts escombrats de la Graella abans i després de l'acció. Després es va variar la funció recompensa creant una finestra d'històrics que tampoc va donar resultat.

Com es veurà en el capítol de resultats, l'entrenament clàssic no ha donat resultat.

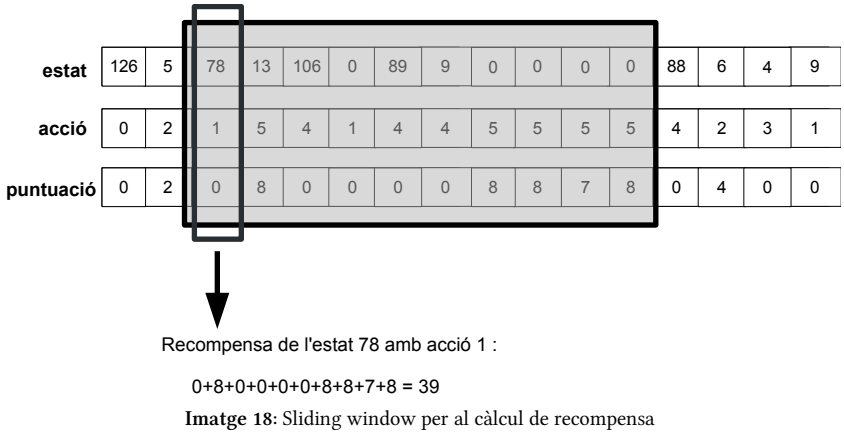
7.5.2. Entrenament ad hoc

L'esperit de l'algorisme de l'aprenentatge per reforç és permetre que un controlador nou, aprengui per sí mateixa a través de la seva experiència. A més, si es pot es busca que sigui la propi controlador el que s'entreni a ella mateixa. En el moment de plantejament del disseny de les proves simulades, s'ha seguit aquest esperit. L'entrenament del controlador per aprenentatge per reforç simula el procés que un robot netejador real que cada dia surt de la seva base i durant 500 unitats temporals, explora i aprèn.

El procés d'aprenentatge és la modificació de la matriu Q a través de l'equació 10. En l'aplicació d'aquest algorisme, s'han provat diversos valors dels paràmetres α i γ .

La creació de la funció recompensa ha requerit un estudi de determinació per a una bona recompensa ja que el sistema usat per a quantificar la superfície escombrada pel robot (la puntuació de la Graella de punts) és independent dels estats triats per a l'aprenentatge per reforç. Per tant un estat més una acció no és recompensat sempre de la mateixa manera.

La recompensa d'una acció ha sigut calculada de dues maneres diferents. En un primer moment s'ha pres la recompensa com els punts escombrats per l'acció que provoca el canvi d'estat, és a dir, la diferència entre els punts escombrats de la Graella abans i després de l'acció. Aquesta funció de recompensa comportava l'inconvenient de que per a una mateix estat-acció, la puntuació no era la mateixa en tots els casos. S'ha pensat i provat una segona funció de recompensa (Imatge 18) ha consistit en crear una finestra històrics de varis estat-acció-puntuació i calcular la recompensa d'una acció com el sumatori de les puntuacions d'aquella acció i les següents.

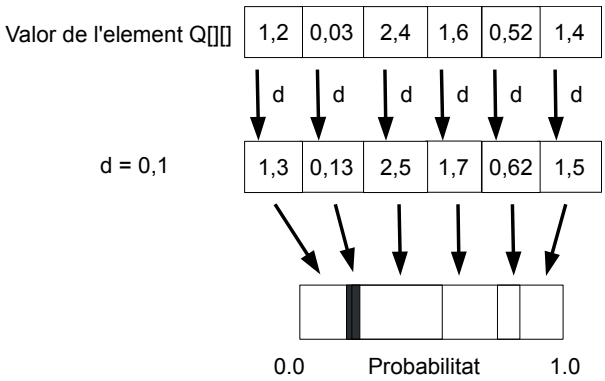


Imatge 18: Sliding window per al càlcul de recompensa

Per a l'entrenament del controlador per aprenentatge per reforç, s'ha dissenyat una funció de decisió basada en la matriu Q. Per a l'estat actual, es pren el valor de cada acció de l'estat actual (Q[estat][i] per i=0..5) i se'ls suma un valor delta. Amb aquests valors nous s'estableixen uns percentatges respecte el total. Aleshores es pren una variable aleatòria entre 0 i 1 i s'identifica en el percentatge de quina acció ha caigut i per tant, quina acció és seleccionada. D'aquesta manera, el controlador comença amb una exploració aleatòria (doncs la matriu Q[][] és zero per a tots els elements) i poc a poc va variant les seves tendències per a seguir aquelles accions més avantatjoses i així consolidar el coneixement i convergir de manera més ràpida.

El gran avantatge d'aquest sistema es que el controlador del robot aprèn mentre treballa i alhora s'especialitza en l'entorn on es fa servir.

El fet de tenir aleatorietat garanteix a llarg termini l'exploració d'accions descartades, de tal manera que si el controlador aprengués un comportament que no és l'òptim (per exemple girar al detectar una paret que és lluny) amb aquest sistema és capaç de corregir el comportament amb un altre comportament més beneficiós.



Imatge 19: Distribució probabilística d'una nova acció per a un estat determinat durant l'entrenament

8. Resultats

En aquest apartat s'engloben els resultats del projecte de tesi.

8.1. El simulador

El primer resultat d'aquest projecte és la creació del simulador. Aquesta eina ha sigut generada per aquest projecte fruit de no trobar una eina amb el grau de flexibilitat i ductilitat.

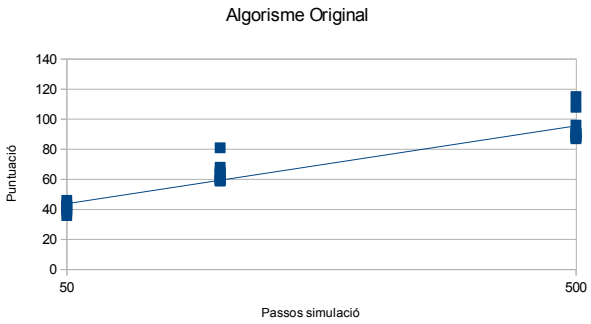
El projecte de Simulador, ha acomplert les expectatives de balanceig entre velocitat i precisió .

8.2. Resultats experimentals obtinguts

En aquest apartat es recullen les dades i gràfiques que resumeixen les execucions realitzades.

8.2.1. Simulació de referència amb l'Algorisme original

El aquesta secció es veu la representació de les puntuacions de 10 simulacions usant l'algorisme reproduït del robot original amb temps 50 passos, 100 passos i 500 passos. El banc de proves és Entrenament.



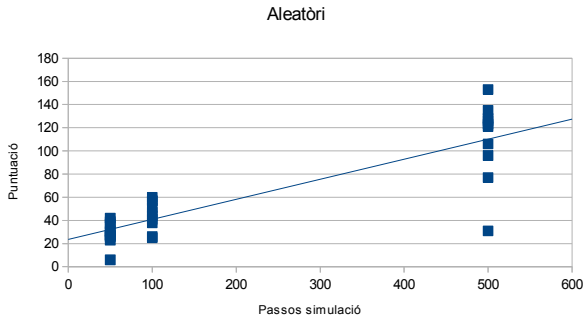
Imatge 20: Simulació de l'algorisme original

Passos	Mitjana	Desviació estàndard
50	40,9	2,85
100	63,8	6,78
500	94,3	9,53

Taula 3: Mitjana i desviació estàndard

8.2.2. Simulació de referència amb el controlador aleatori

El aquesta secció es veu la representació de les puntuacions de 10 simulacions usant el controlador de comportament aleatori descrita a la secció 7.3 Controlador amb comportament aleatori: IARandom amb temps 50 passos, 100 passos i 500 passos. El banc de proves és Entrenament.



Imatge 21: Simulació amb aleatòria

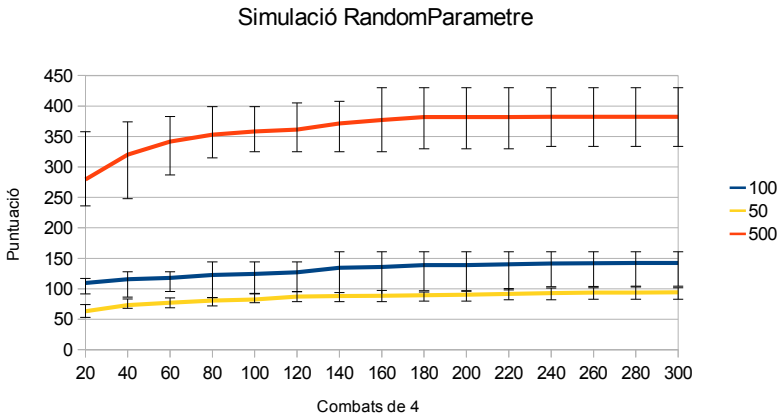
Passos	Mitjana	Desviació estàndard
50	28,8	10,1083024182
100	44,6	12,3756032401
500	109,6	34,7761092962

Taula 4: Mitjana i desviació estàndard

8.2.3. Simulació del RandomParametre

El aquesta secció es veu la representació de les puntuacions màximes assolides en el pas de les generacions de 10 simulacions independents de poblacions usant el controlador per algorismes genètics a la secció 7.4.1 (Comportament aleatori amb probabilitat fixada per gens i paràmetres de moviment fixats: IARandomParametre) amb temps 50 passos, 100 passos i 500 passos. El banc de proves és Entrenament. La població inicial és de 50 individus i es realitzen 300 noves generacions. El mètode d'obtenció d'una nova generació és el combat de 4 participants.

En l'eix de les X es representa el nombre de generacions noves creades, l'eix de les Y representa la puntuació màxima obtinguda a la població. Per tant en aquest gràfic es pot veure l'evolució de la puntuació màxima d'una població al llarg de les generacions. La funció representada és la mitja de les 10 puntuacions per a cada nombre de combats fixat. A més, es reflecteixen els valors mínims i màxims de la serie de 10 puntuacions de les quals es fa la mitja. El resultat es pot veure en la Imatge 22.



Imatge 22: Gràfica de la mitja de les 10 simulacions amb el valor màxim i mínim

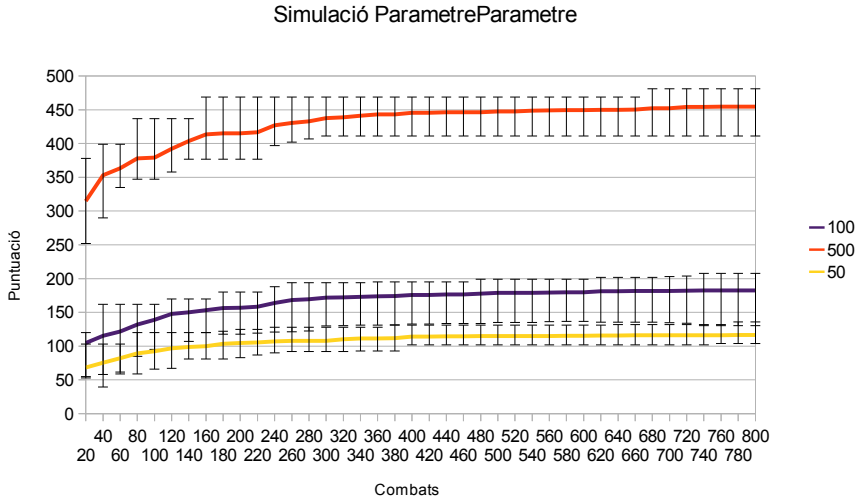
		20	40	60	80	100	120	140	160	180	200	220	240	260	280	300
50	mitja	63,3	73,1	77	80,5	82,4	87,4	88,1	88,6	89,3	90,5	91,7	92,8	94	94	94,3
	deltaM	10,7	7,9	9	5,5	5,6	8,6	7,9	7,4	6,7	5,5	8,3	7,2	9	9	9,7
	deltam	10,3	5,1	8	8,5	5,4	8,4	9,1	9,6	9,3	10,5	9,7	10,8	11	11	11,3
	Desv.	7,29	4,25	5,68	5,17	4,17	4,74	5,47	5,78	5,48	5,62	6,06	5,71	6,65	6,65	6,86
100	mitja	109,4	115,5	117,8	122,6	124,5	127,1	134,6	136	138,8	138,8	140,1	141,6	141,8	142,3	142,3
	deltaM	7,6	12,5	10,2	21,4	19,5	16,9	26,4	25	22,2	22,2	20,9	19,4	19,2	18,7	18,7
	deltam	18	29	22	37	32	32	46	46	42	42	42	38	38	38	38
	Desv.	5,46	9,5	7,57	11	11,76	12,29	14,7	13,94	13,41	13,41	13,3	11,36	11,49	11,5	11,5
500	mitja	279,6	320,2	341,9	353,1	358,4	361,4	371,4	377,2	381,9	382	382	382,4	382,5	382,5	382,5
	deltaM	78,4	53,8	41,1	45,9	40,6	43,6	36,6	52,8	48,1	48	48	47,6	47,5	47,5	47,5
	deltam	43,6	72,2	54,9	38,1	33,4	36,4	46,4	52,2	51,9	52	52	48,4	48,5	48,5	48,5
	Desv.	38,14	38,62	33,78	25,97	25,59	27,19	28,06	28,99	31,65	31,71	31,71	31	31,06	31,06	31,06

Taula 5: Mitjanes, desviacions estàndards i deltes del màxim i mínim

8.2.4. Simulació del ParametreParametre

El aquesta secció es veu la representació de les puntuacions màximes assolides en el pas de les generacions de 10 simulacions independents de poblacions usant l'algorisme genètic de la secció 7.4.2 (Comportament aleatori amb probabilitat fixada per gens i paràmetres de moviment per gens: IAParametreParametre) amb temps 50 passos, 100 passos i 500 passos. El banc de proves és Entrenament. La població inicial és de 100 individus i es realitzen 800 noves generacions. El mètode d'obtenció d'una nova generació és el combat de 4 participants.

En l'eix de les X es representa el nombre de generacions noves creades, l'eix de les Y representa la puntuació màxima obtinguda a la població. Per tant en aquest gràfic es pot veure l'evolució de la puntuació màxima d'una població al llarg de les generacions. La funció representada és la mitja de les 10 puntuacions per a cada nombre de combats fixat. A més, es reflecteixen els valors mínims i màxims de la serie de 10 puntuacions de les quals es fa la mitja. El resultat es pot veure en la Imatge 23.



Imatge 23: Gràfica de la mitja de les 10 simulacions amb el valor màxim i mínim

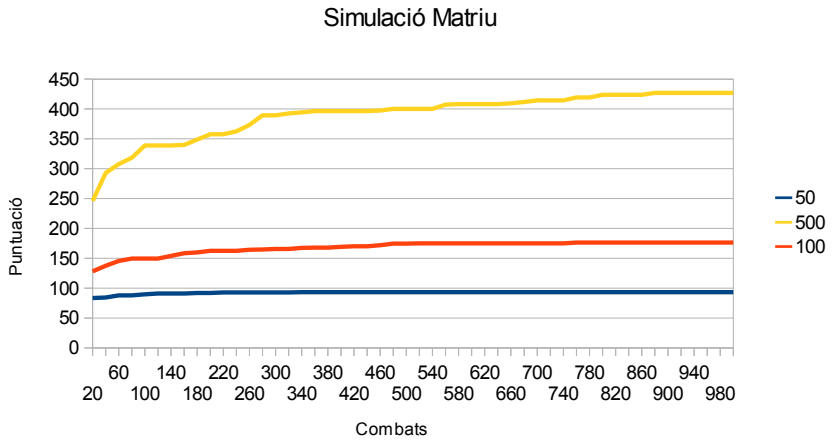
		100	200	300	400	500	600	700	800
50	Mitja	92,4	104,9	108	114,2	115,1	115,4	116	116,6
	Desv.	16,04	14,32	11,93	9,31	8,84	9,09	9,3	9,81
100	Mitja	139,1	156,7	172	175,8	179,1	179,6	181,9	182,4
	Desv.	17,25	11,71	13,3	14,76	14,3	13,91	15,3	16,13
500	Mitja	379,2	415,3	437,7	445,7	447,7	449,7	452,4	454,8
	Desv.	27,2	27,21	17,31	20,98	20,06	18,87	20,84	21,67

Taula 6: Mitjanes i desviacions estàndard per nombre de generacions

8.2.5. Simulació del Matriu de 3 accions

El aquesta secció es veu la representació de les puntuacions màximes assolides en el pas de les generacions de 10 simulacions independents de poblacions usant l'algorisme genètic de la secció 7.4.3. (Comportament calculat segons els gens i els sensors: IAMatriu) amb temps 50 passos, 100 passos i 500 passos. El banc de proves és Entrenament. La població inicial és de 100 individus i es realitzen 1000 noves generacions. El mètode d'obtenció d'una nova generació és el combat de 4 participants.

En l'eix de les X es representa el nombre de generacions noves creades, l'eix de les Y representa la puntuació màxima obtinguda a la població. Per tant en aquest gràfic es pot veure l'evolució de la puntuació màxima d'una població al llarg de les generacions. La funció representada és la mitja de les 10 puntuacions per a cada nombre de combats fixat. A més, es reflecteixen els valors mínims i màxims de la serie de 10 puntuacions de les quals es fa la mitja. El resultat es pot veure en la Imatge 24.



Imatge 24: Gràfica de les mitges de 10 simulacions

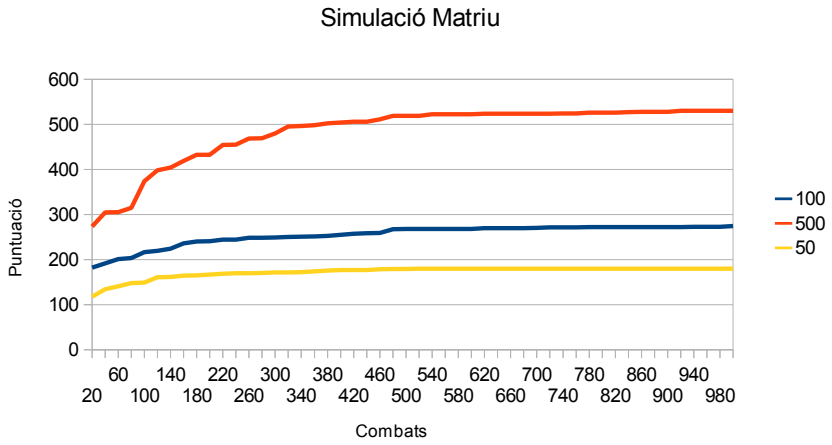
		100	200	300	400	500	600	700	800	900	1000
50	Mitja	89,7	91,9	92,8	93	93	93	93	93	93	93
	Desv.	5,31	3,96	1,99	2,21	2,21	2,21	2,21	2,21	2,21	2,21
100	Mitja	149,4	162,3	165,7	169	174,5	174,8	174,9	176,2	176,2	176,2
	Desv.	15,23	7,6	8,26	6,9	5,6	5,41	5,38	5,83	5,83	5,83
500	Mitja	338,8	357,6	389,5	396,4	399,9	408	414,6	423,8	426,8	427,1
	Desv.	45,11	47,55	49,69	46,45	44,09	35,29	37,66	40,11	38,82	38,73

Taula 7: Mitges i desviacions de les 10 simulacions de IAMatriu

8.2.6. Simulació del Matriu de 6 accions

El aquesta secció es veu la representació de les puntuacions màximes assolides en el pas de les generacions de 10 simulacions independents de poblacions usant l'algorisme genètic de la secció 7.4.3. (Comportament calculat segons els gens i els sensors: IAMatriu) amb temps 50 passos, 100 passos i 500 passos. El banc de proves és Entrenament. La població inicial és de 400 individus i es realitzen 1000 noves generacions. El mètode d'obtenció d'una nova generació és el combat de 4 participants.

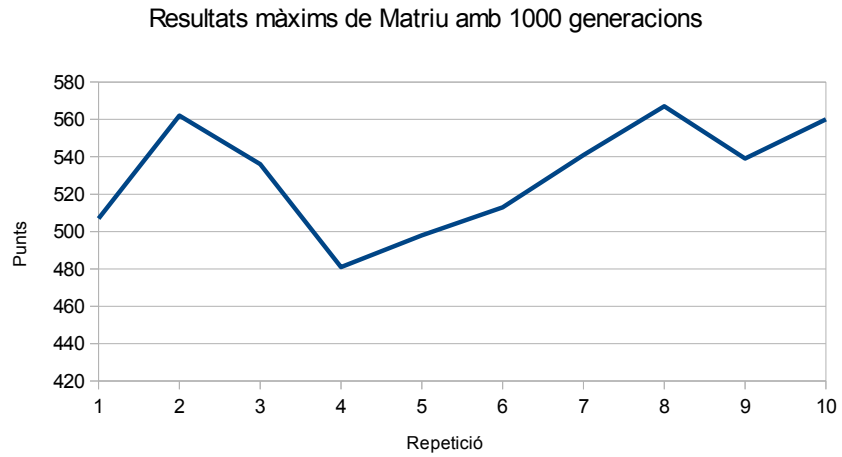
En l'eix de les X es representa el nombre de generacions noves creades, l'eix de les Y representa la puntuació màxima obtinguda a la població. Per tant en aquest gràfic es pot veure l'evolució de la puntuació màxima d'una població al llarg de les generacions. La funció representada és la mitja de les 10 puntuacions per a cada nombre de combats fixat. A més, es reflecteixen els valors mínims i màxims de la serie de 10 puntuacions de les quals es fa la mitja. El resultat es pot veure en la Imatge 25.



Imatge 25: Gràfica de la mitja de les 10 simulacions

		100	200	300	400	500	600	700	800	900	1000
50	Mitja	149,2	167	171,9	176,7	179,1	180,1	180,1	180,1	180,1	180,1
	Desv.	18,84	18,6	18,02	12,12	7,17	7,08	7,08	7,08	7,08	7,08
	Max	172	184	187	187	187	187	187	187	187	187
	Min	110	122	130	148	172	172	172	172	172	172
100	Mitja	216,4	240,7	249,1	255	267,9	268	270,3	272,4	272,4	274,7
	Desv.	25,56	18,94	12,74	10,65	19,13	19,15	20,18	22,74	22,74	23,72
	Max	252	280	280	280	311	311	311	311	311	311
	Min	175	211	233	244	247	247	247	247	247	247
500	Mitja	373,9	432,6	479,9	504	519,2	522,6	523,8	526	527,8	530,4
	Desv.	61,07	46,19	37,8	42,36	32,49	32,3	30,6	28,45	27,79	29,39
	Max	455	499	538	556	562	562	562	562	567	567
	Min	247	366	414	419	473	473	477	477	480	481

Taula 8: Mitja, desviació estàndard i màxims i mínims

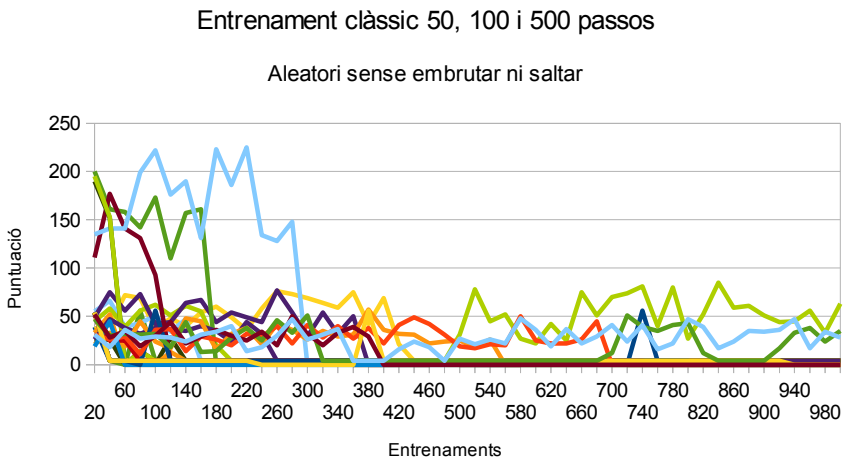


Imatge 26: Simulació de 10 repeticions dels IndividuMatriu (6 accions) campions amb 1000 generacions

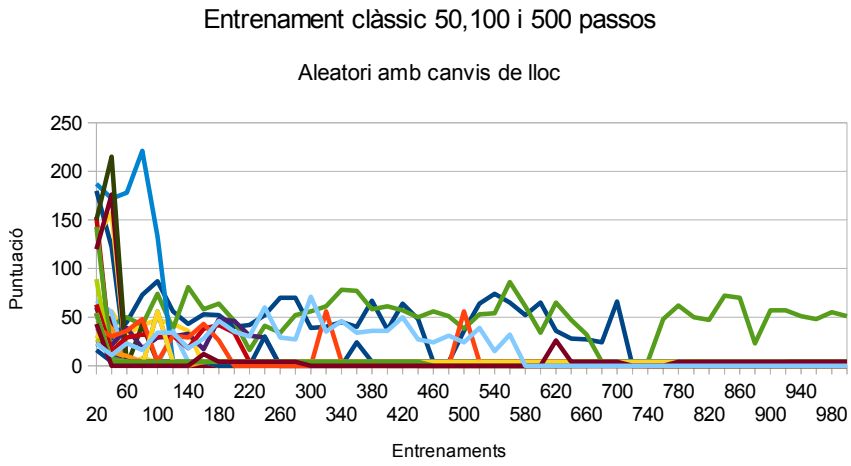
8.2.7. Simulació del RL

El aquesta secció es veuen les gràfiques comparatives de les simulacions realitzades amb usant l'algorisme per aprenentatge de reforç descrit a la secció 7.5. (Controlador per Reinforcement Learning).

8.2.7.1. Mètode clàssic



Imatge 27: Entrenament clàssic aleatori finestra de 1



Imatge 28: Entrenament clàssic aleatori amb salts de posició, finestra de 1

8.2.7.2. Mètode adhoc

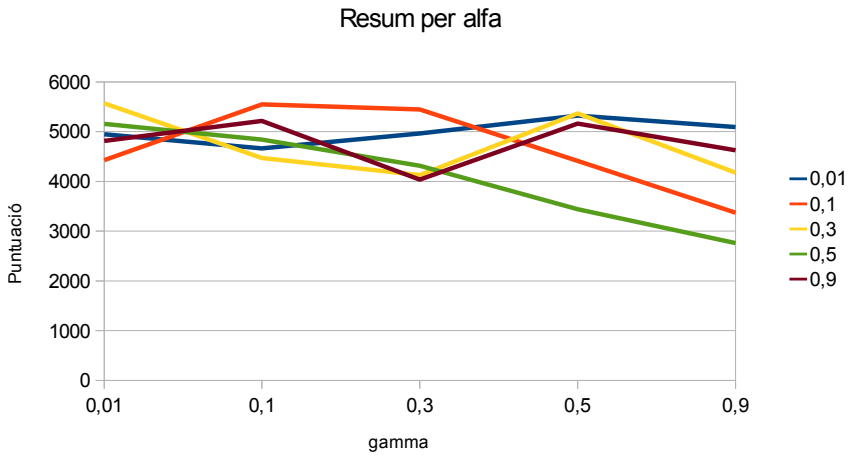
8.2.7.2.1. Exploració α i γ per 4096 estats

Per a simular aquest controlador, s'ha simulat un robot de 4096 estats que durant 14 dies surt de la base i disposa de 500 passos de temps (abans d'esgotar la bateria) per a escombrar tot el que pugui. El robot, al primer dia no té cap coneixement, no té comportaments apresos. A mesura que va interactuant i escombrant, aprèn (actualitza la Q). S'ha simulat els 14 dies d'aprenentatge amb diversos paràmetres de alfa, gamma, delta i mida de la finestra, i els resultats dels 14 dies s'han recollir per separat i després sumats.

Comparació de les variables alfa i gamma (prenent les mitges de les simulacions amb permutacions de les altres variables):

alfa\gamma	0,01	0,1	0,3	0,5	0,9	Grand Total
0,01	4949	4664	4965	5325	5094	5325
0,1	4427	5548	5447	4412	3369	5548
0,3	5573	4473	4126	5369	4174	5573
0,5	5158	4840	4313	3440	2757	5158
0,9	4814	5215	4035	5165	4624	5215
Grand Total	5573	5548	5447	5369	5094	5573

Taula 9: Contrast alfa-gamma



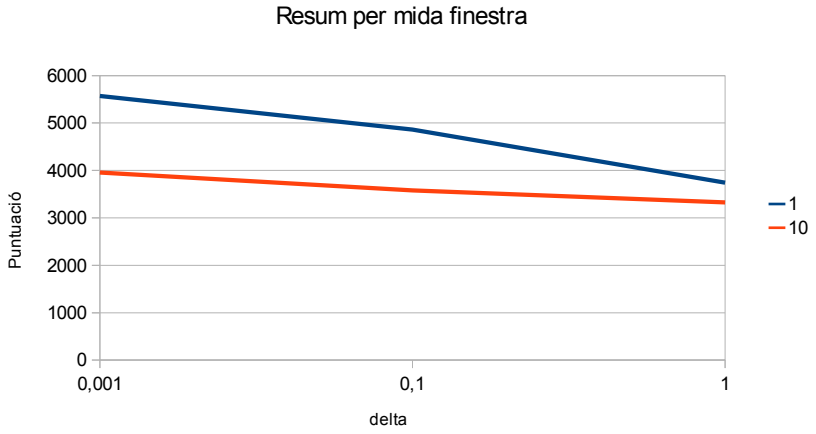
Imatge 29: Puntuacions de les diferents alfes

Comparació de les variables mida de la finestra i delta (prenent les mitges de les simulacions amb permutacions de les altres variables):

Mida finestra \ delta	0,01	0,1	1	Grand Total
1	5573	4860	3741	5573
10	3954	3580	3324	3954
Grand Total	5573	4860	3741	5573

Taula 10: Contrast mida finestra -delta

En la Taula 10 es veu perfectament com fixant delta a 0,01 i la mida de la finestra a 1 s'aconsegueixen els millors resultats.



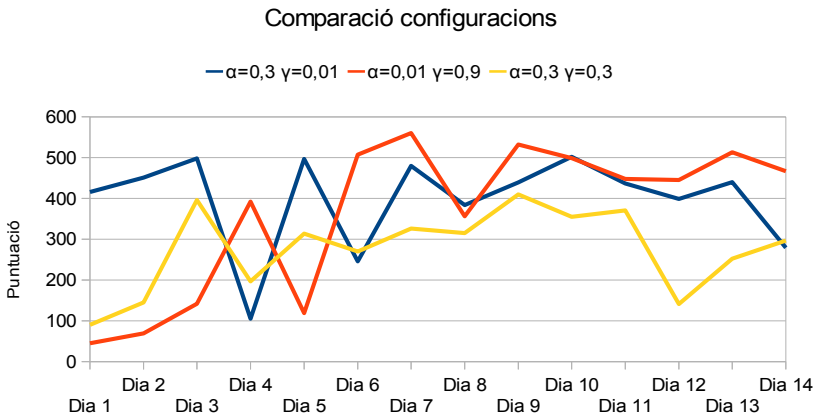
Imatge 30: Puntuacions de les diferents mides de finestra

Donada la gran quantitat de combinacions de valors obtingudes, s'estableix un mètode de selecció dels millors paràmetres basat en la comparació de les puntuacions obtingudes per les simulacions, havent filtrat pels valors màxims. Els valors de delta seran fixats a 0,001 i la mida de la finestra serà fixada a 1.

Alfa \ gamma	0,01	0,1	0,3	0,5	0,9	Grand Total
0,01	4949	4664	4965	5325	5094	5325
0,1	4427	5548	5447	4412	3369	5548
0,3	5573	4444	3879	5369	4174	5573
0,5	5158	4840	4313	3329	2458	5158
0,9	4814	5215	4035	5165	4624	5215
Grand Total	5573	5548	5447	5369	5094	5573

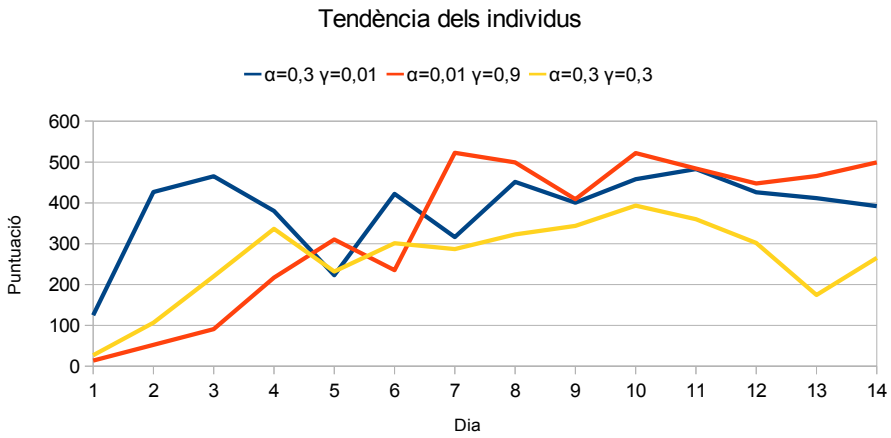
Taula 11: Contrast alfa-gamma corregit

A Taula 11 es veu com la correcció dels valors delta i finestra ha comportat que tres configuracions d'alfa i gamma sobresurtin de la resta. Agafant els resultats de les simulacions per a aquests valors, es comparen les configuracions.



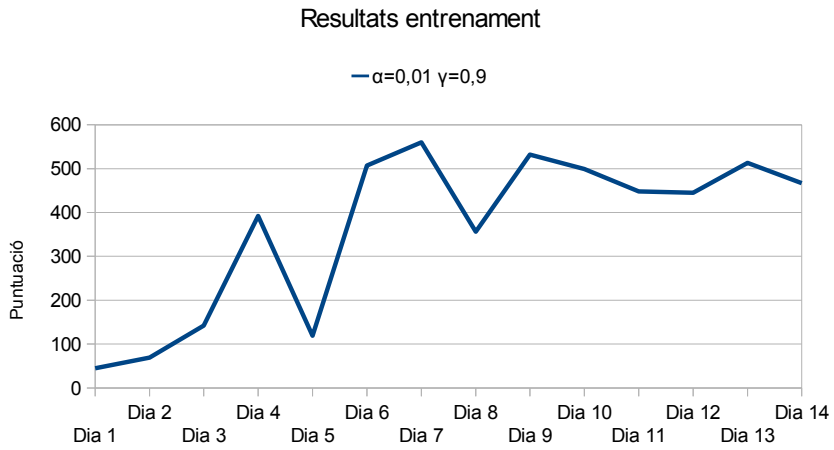
Imatge 31: Puntuació aconseguida per dia

Com es pot veure a la Imatge 31, amb aquestes configuracions el robot va aprenent seguint una corba oscil·lant, és a dir, per a millorar el seu criteri, ha de passar per un procés d'empitjorament transitori. Hem calculat la tendència dels individus com la suma d'un 0,7 pel valor de tendència anterior més un 0,3 pel valor de la puntuació aconseguida en el dia.

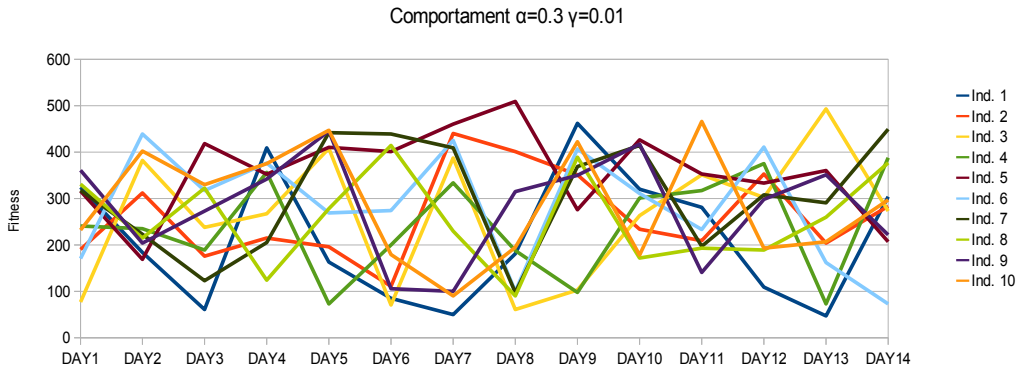


Imatge 32: Tendència

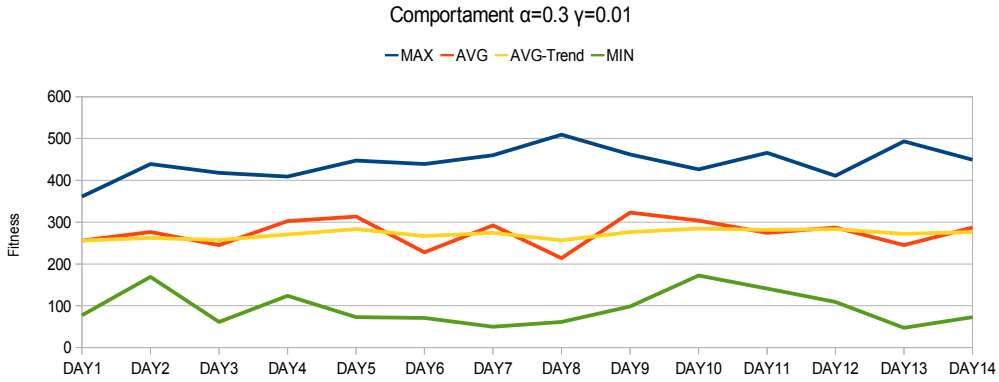
Es pot veure a la Imatge 32 com la tendència de tots tres és incrementar la puntuació. El càlcul de la tendència permet atenuar els moments d'empitjorament transitori i veure l'evolució i l'aprenentatge en termes globals. Segons els resultats obtinguts, seleccionaríem la configuració de alfa 0,01 i gamma 0,9.



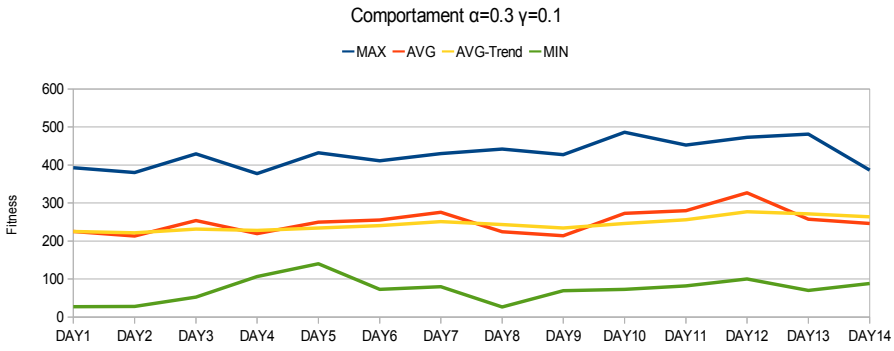
Imatge 33: Resultats entrenament



Imatge 34: Evolució de 10 individus independents amb el millor comportament



Imatge 35: Estadístiques de la Imatge 34

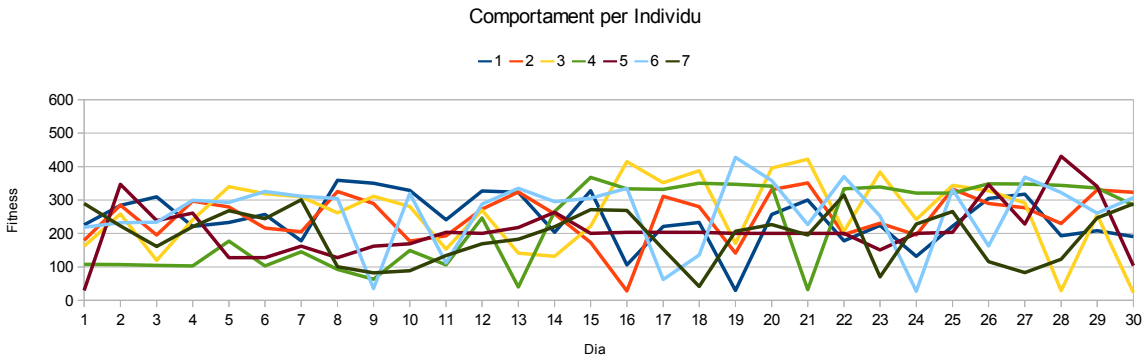


Imatge 36: Estadístiques de 10 individus independents amb el segon millor comportament

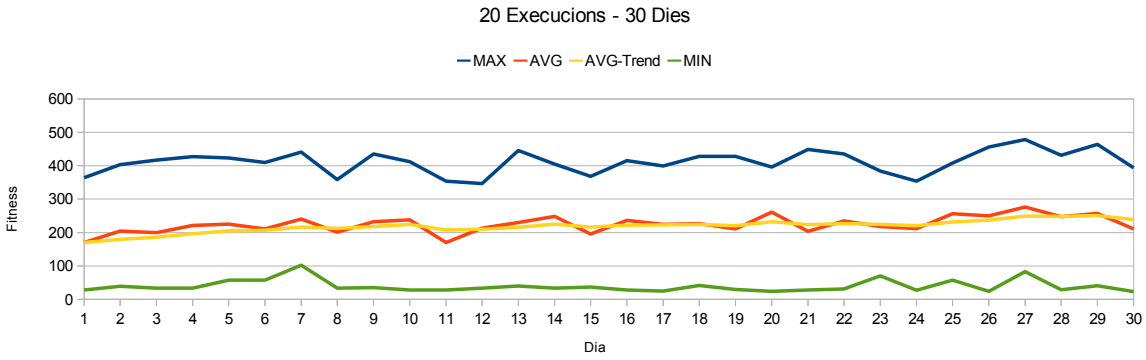
8.2.7.2.2. α i γ variables per a 4096 estats

Per a intentar baixar la variabilitat, s'han dissenyat una sèrie d'experiments que definien alfa i gamma

variables en funció del temps. S'ha comprovat que variacions dels valors de alfa i gamma influeixen en la periodicitat en que el controlador oblida part del que sap per a aprendre més en el següent dia.



Imatge 37: Mostra aleatòria d'individus de la simulació de alfa i gamma variable



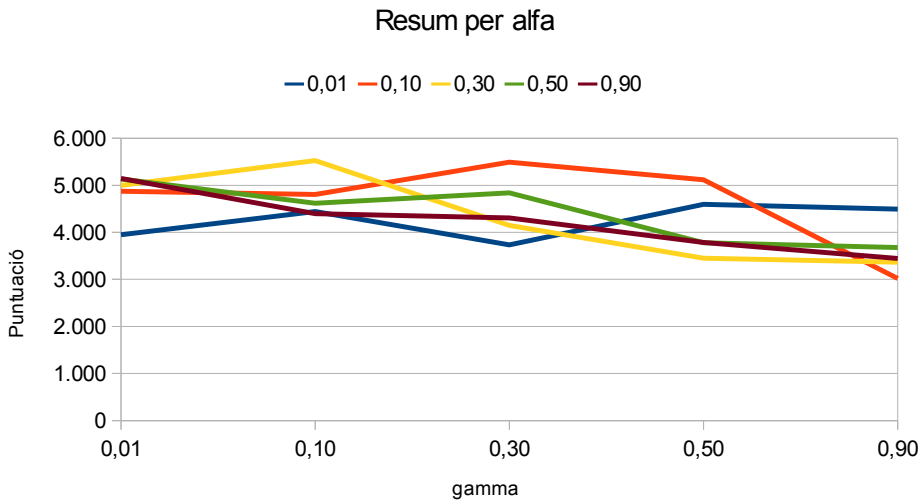
Imatge 38: Estadístiques d'una simulació 20 execucions de 30 dies amb totes les permutacions de alfa i gamma

8.2.7.2.3. Exploració α i γ per 256 estats

Addicionalment s'ha simulat un robot de 256 estats que durant 14 dies surt de la base i disposa de 500 passos de temps (abans d'esgotar la bateria) per a escombrar tot el que pugui. El robot, al primer dia no té cap coneixement, no té comportaments apresos. A mesura que va interactuant i escombrant, aprèn (actualitza la Q). S'ha simulat els 14 dies d'aprenentatge amb diversos paràmetres de alfa, gamma, delta i mida de la finestra, i els resultats dels 14 dies s'han recollir per separat i després sumats.
Comparació de les variables alfa i gamma (prenent les mitges de les simulacions amb permutacions de les altres variables):

0,00	0,01	0,10	0,30	0,50	0,90
0,01	3.950	4.442	3.736	4.595	4.492
0,10	4.872	4.807	5.494	5.114	3.018
0,30	4.996	5.523	4.150	3.448	3.364
0,50	5.132	4.616	4.839	3.781	3.675
0,90	5.145	4.394	4.305	3.785	3.442

Taula 12: Contrast alfa i gamma

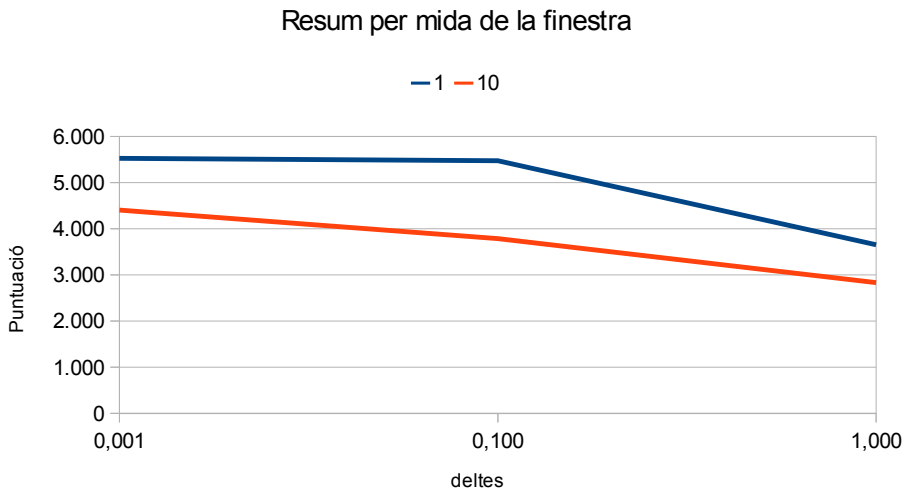


Imatge 39: Contrast per alfa

Comparació de les variables mida de la finestra i delta (prenent les mitges de les simulacions amb permutacions de les altres variables):

mida finestra \ delta	0,001	0,100	1,000
1	5.523	5.472	3.653
10	4.404	3.788	2.831

Taula 13: Contrast mida de la finestra i delta



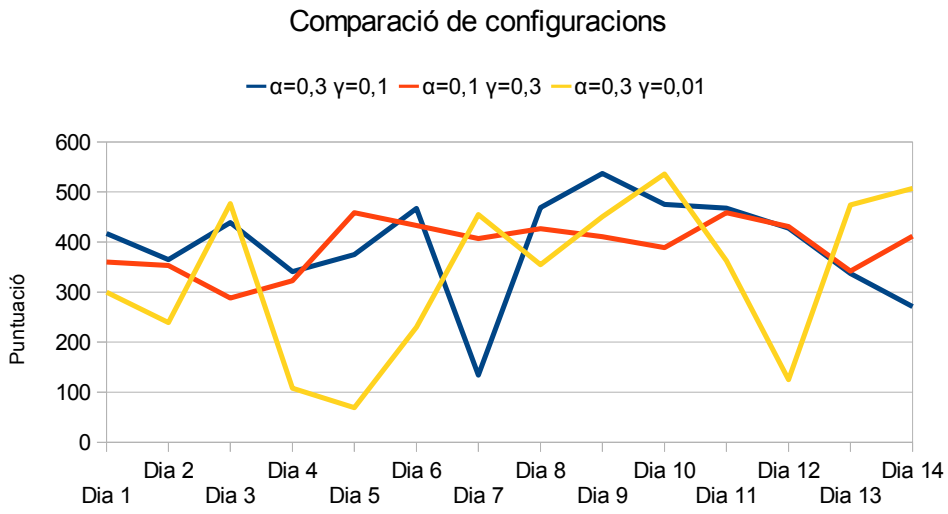
Imatge 40: Puntuacions de les diferents mides de finestra

Com en el cas anterior de 4096 estats, donada la gran quantitat de combinacions de valors s'ha d'establir un mètode de selecció dels millors paràmetres basat en la comparació de les puntuacions obtingudes per les simulacions, havent filtrat pels valors màxims. Els valors de delta seran fixats a 0,001 i la mida de la finestra serà fixada a 1.

Alfa\gamma	0,01	0,10	0,30	0,50	0,90	Grand Total
0,01	3.369	3.139	3.422	3.941	4.492	4.492
0,10	4.327	4.807	5.494	5.114	2.762	5.494
0,30	4.689	5.523	4.150	3.172	3.049	5.523
0,50	5.132	4.616	4.839	3.781	2.545	5.132
0,90	5.145	4.248	4.305	3.785	3.442	5.145
Grand Total	5.145	5.523	5.494	5.114	4.492	5.523

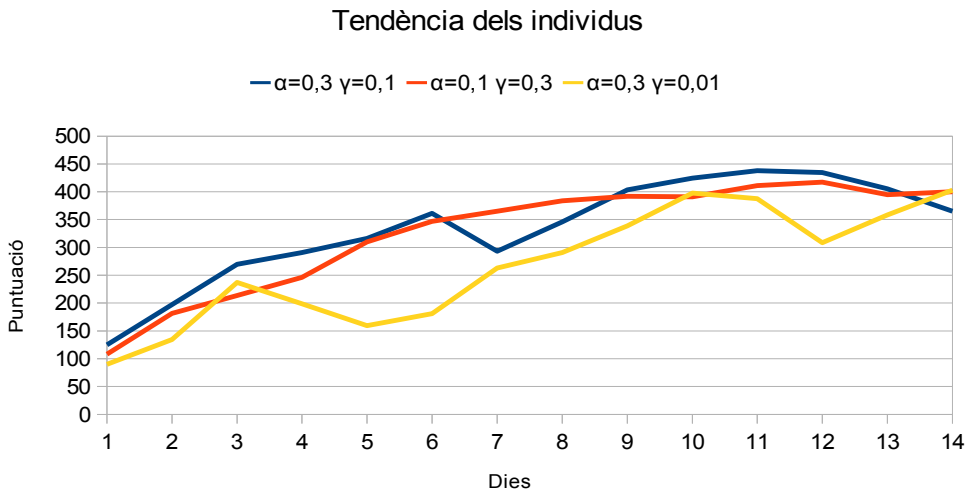
Taula 14: Contrast d'alfa i gamma amb valors corregits

A Taula 14 es veu com la correcció dels valors delta i finestra ha comportat que dos configuracions d'alfa i gamma sobresurtin de la resta. Agafant els resultats de les simulacions per a aquests valors i afegint la configuració amb resultat més alt de la simulació de 4096 estats (alfa 0,3 i gamma 0,01), es comparen les configuracions.



Imatge 41: Comparació de configuracions

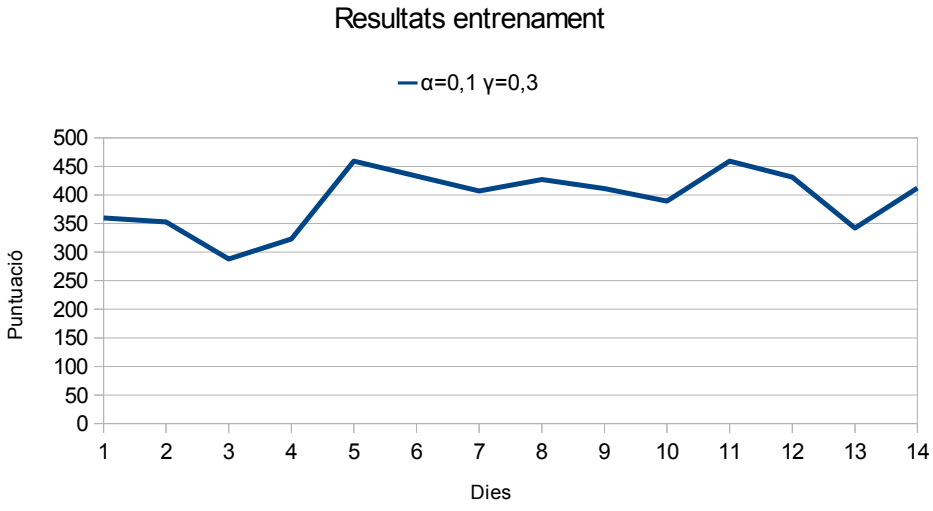
Tal i com mostra la Imatge 41, existeixen oscil·lacions en la puntuació i no es pot apreciar a cap tendència, per tant cal calcular-les. Les tendències corresponents a aquestes configuracions són:



Imatge 42: Tendències

Com es pot veure a la Imatge 42, s'ha afegit a les configuracions millors de les simulacions de 256 estats, la millor configuració de la simulació de 4096 estats (alfa 0,3 i gamma 0,01). La configuració seleccionada seria alfa 0,1 i gamma 0,3.

Totes tres configuracions mostren una tendència creixent i en comparació a la simulació de 4096 estats, sense gaires oscil·lacions. Tanmateix en el cas de 4096 estats, les puntuacions finals de la tendència als 14 dies eren més altes.



Imatge 43: Resultats entrenament robot de 256 estats

8.3. Comparació dels diferents controladors

Amb els resultats obtinguts, es realitzen comparacions d'eficàcia amb els controladors creats.

8.3.1. Els navegadors de referència IAOctagon2 i IARandom

La primera comparació es fa entre els navegadors creats com a referències, és a dir, l'algorisme de l'ASTAN seguint el manual d'usuari (IAOctagon2) i el navegador de referència aleatori amb tres accions (IARandom). Tal i com es pot apreciar a la Taula 3 i la Taula 4, els seus resultats són similars, sent superior l'algorisme aleatori per a execucions amb 50 passos i 500 passos, i superior l'ASTAN en l'execució de 100 passos.

Els resultats extrets en ambdós navegadors són baixos. El resultat del navegador original es pot assumir que serà pitjor que el real perquè el IAOctagon2 està basat en unes afirmacions del manual que podrien no ser exactes. Els resultats baixos de l'algorisme aleatori poden estar basats en que el navegador només realitza tres accions, dues de les quals no escombren i per tant no incrementa puntuació.

8.3.2. El genètic IARandomParametre comparat amb el IARandom

El navegador IARandom i el navegador IARandomParametre són molt similars: ambdós navegadors només són capaços de decidir entre tres accions, les quals corresponen a les primeres tres accions de CadenaInput. A més, tant IARandom com IARandomParametre usen una variable probabilística per a decidir quina nova acció efectuaran i cap dels dos navegadors rep cap informació dels sensors.

La diferència que separa al navegador IARandom del IARandomParametre és la distribució de la probabilitat de les tres accions. Mentre que a IARandom manté una probabilitat d'un terç a cada acció, a IARandomParametre la probabilitat de cada acció ve determinada pel codi genètic total, tal i com s'ha vist a l'apartat 7.4.1.

En un inici, la població de IARandomParametres (població inicial de 50 individus) es genera amb gens d'una distribució uniforme. Per tant, en la mitja del codi genètic de la població inicial de IARandomParametres determinaria un individu una probabilitat d'un terç a cada acció. A l'origen, IARandom i el conjunt dels IARandomParametres tenen comportaments idèntics.

Per a mesurar les virtuts i defectes de la determinació de probabilitats per gens, es comparen les simulacions realitzades a l'IARandom (8.2.3.) i a l'IARandomParametre (8.2.4.), totes dues usant el banc de proves Entrenament.

Tal i com recullen les taules 4 i 5, en acabat de les primeres 20 generacions de l'IARandomParametre es dóna una gran distància entre les mitjanes de l'IARandom i el resultat del campió de la població dels individus IARandomParametre, per a 50 passos, l'aleatori fa 28,8 punts (mitjana) enfront als 63,3 punts (en mitjana) dels campions de IARandomParametre. En els 100 passos, l'aleatori fa una puntuació de 44,6 enfront als 109,4 punts dels campions IARandomParametre. En els 500 passos, l'aleatori fa una puntuació de 109,6 enfront als 279,6. Les poblacions de IARandomParametre continuen evolucionant fins a arribar a una estabilització que correspon a uns campions de mitjana 94,3 punts per a 50 passos, 142,3 punts per a 100 passos i 382,5 punts per a 500 passos. Aquests valors màxims són prou bons tot i que els punts màxims a escombrar per al banc de proves Entrenament és 743 i per tant, és un resultat millorable amb un altre navegador.

En conclusió, el sistema de gens com a determinant de l'elecció d'una acció o una altra permet al sistema arribar a bones configuracions de solucions.

8.3.3. El genètic IAParametreParametre comparat amb el IARandomParametre

El navegador IAParametreParametre i el navegador IARandomParametre són molt similars: ambdós navegadors usen una variable probabilística per a decidir entre tres accions i cap dels dos navegadors rep cap informació dels sensors.

La diferència entre el navegador IAParametreParametre del IARandomParametre són les accions. Les accions de IARandomParametre són les primeres accions de CadenaAccio. En canvi cada IAParametreParametre només pot fer tres accions definides en els seus arguments pels seus gens, com s'ha exposat en el capítol 7.4.2. Comportament aleatori amb probabilitat fixada per gens i paràmetres de moviment per gens: IAParametreParametre.

En un inici, la població de IAParametresParametres es genera amb gens d'una distribució uniforme. Com s'ha exposat en el capítol 7.1.2. CadenaAccions, les tres primeres accions corresponen a sengles mitges d'una distribució uniforme entre 0 i l'argument de les tres últimes accions. Com que els gens de IAParametreParametre es generen a partir d'una distribució uniforme, la mitja de cadascuna coincideix amb l'acció corresponent de les tres primeres accions. Per tant, en un inici, les poblacions de IARandomParametre i IAParametreParametre tenen les accions amb idèntica mitja.

Per a comparar l'aventatge o desavantatge de determinar les accions per gens, es comparen les simulacions realitzades amb IARandomParametre (Imatge 22) amb les simulacions de IAParametreParametre (Imatge 23). Les poblacions inicials de IARandomParametre són de 50 individus i les de IAParametreParametre són de 100 individus. Les dues simulacions fan servir el banc de proves Entrenament.

Tal i com recullen les taules 5 i 6, en acabat de les primeres 300 generacions, l'IARandomParametre s'ha estabilitzat completament amb una mitja de 94,3 punts per als 50 passos, 142,3 punts per a 100 passos i 382,5 punts a 500 passos. En canvi l'IAParametreParametre s'ha estabilitzat molt i tot i que no estancat i ha assolit mitges de 108 punts per a 50 passos, 172 punts per a 100 passos i 437,7 punts per a 500 passos. Finalment en les 800 generacions, l'IAParametreParametre ha assolit unes mitges de 116,6 punts per als 50 passos, 182,4 punts per als 100 passos i 454,8 punts per als 500 passos, encara en millora lenta i contínua.

Per tant, es demostra que la determinació dels arguments de les accions mitjançant gens és una millora i que les poblacions de l'IAParametreParametre assoleixen campions amb bons resultats en uns centenars de generacions, és a dir, té una convergència ràpida.

Aquests valors màxims són prou bons tot i que els punts màxims a escombrar per al banc de proves Entrenament és 743 i per tant, encara es pot trobar un algorisme millor.

En conclusió, el sistema de gens com a determinant dels arguments de tres úniques accions permet al sistema arribar a bones configuracions de solucions.

8.3.4. El genètic IARandomParametre comparat amb el gènetic IAMatriu

El navegador IARandomParametre i el navegador IAMatriu tenen alguns punts en comú. El IARandomParametre tria entre 3 accions (les tres primeres accions de CadenaAccio, que són les de menys argument) i el IAMatriu tria entre les mateixes tres accions.

L'algorisme de decisió de IARandomParametre usa una variable probabilística i els seus gens per a decidir l'acció i en canvi l'algorisme de decisió de l'IAMatriu usa els valors dels sensors del robot i els seus gens, tal com s'ha explicat en l'apartat 7.4.3. Comportament calculat segons els gens i els sensors: IAMatriu.

Per a comparar l'avantatge o desavantatge de determinar les accions pels sensors, es comparen les simulacions realitzades amb IARandomParametre (Imatge 22) amb les simulacions de IAMatriu (Imatge 24). Les poblacions inicials de IARandomParametre són de 50 individus i les de IAMatriu són de 100 individus. Les dues simulacions fan servir el banc de proves Entrenament.

Tal i com recullen les taules 5 i 7, en acabat de les primeres 300 generacions, l'IARandomParametre s'ha estabilitzat completament amb una mitja de 94,3 punts per als 50 passos, 142,3 punts per a 100 passos i 382,5 punts a 500 passos. En canvi la IAMatriu en 300 generacions ha assolit una mitja de 92,8 punts per a 50 passos, 167,5 punts per a 100 passos i 389,5 punts per a 500 passos, és a dir, IAMatriu és pitjor que IARandomParametre en pocs passos però és millor en execucions més llargues. Finalment en les 1000 generacions, l'IAMatriu ha assolit unes mitges de 93 punts per als 50 passos, 176,2 punts per als 100 passos i 427,1 punts per als 500 passos, és a dir, continua sent millor de IARandom per a simulacions llargues però és pitjor en les curtes.

Per tant, es determina que la determinació de l'acció a través de càlculs de gens usant els sensors és una millora en execucions de 100 passos o més, però per a simulacions més curtes, té un comportament similar a un aleatori.

En conclusió, el sistema de gens i sensors com a selector de les tres úniques accions permet al sistema millorar els resultats de l'aleatori si l'execució és prou llarga.

8.3.5. El genètic IAParametreParametre comparat amb el genètic IAMatriu

El navegador IAParametreParametre i el navegador IAMatriu són molt diferents ja que no comparteixen cap criteri ni algorisme de decisió. Tot i així, IAParametreParametre és el millor navegador dels genètics sense usar les lectures dels sensors, per tant, és procedent comparar l'eficàcia del navegador IAMatriu i el IAParametreParametre.

Les poblacions inicials de IAMatriu són de 400 individus i les de IAParametreParametre són de 100 individus. Les dues simulacions fan servir el banc de proves Entrenament.

Tal i com recullen les taules 6 i 8, en acabat de les primeres 800 generacions, l'IAParametreParametre ha assolit b una mitja de 116,6 punts per als 50 passos, 182,4 punts per a 100 passos i 454,8 punts a 500 passos. En canvi l'IAMatriu ha ha assolit una mitja de 180,1 punts per a 50 passos, 272,4 punts per a 100 passos i 526 punts per a 500 passos. Finalment en les 1000 generacions, l'IAMatriu ha assolit unes mitges de 180,1 punts per a 50 passos, 274,7 punts per a 100 passos i 530,4 punts per a 500 passos, que indiquen una estabilització. Aquests valors màxims són prou bons però encara es pot trobar un algorisme millor.

Es demostra que el controlador IAMatriu és el millor de tots els genètics i que la combinació de gens i sensors com a determinants de les accions permet al sistema arribar a bones configuracions de solucions.

8.3.6. El genètic IAMatriu comparat a l'aprenentatge per reforç IARL

El navegador IAMatriu i el navegador IARL són molt similars: ambdós navegadors usen els valors dels sensors per a escollir l'acció a realitzar.

En el moment del disseny dels controladors i les seves capacitats, es va cercar crear dos navegadors que usant diferents tècniques (un els algorismes genètics i l'altre l'aprenentatge per reforç) fossin capaços de a partir de les mateixes variables d'entrada, decidir una acció de les (mateixes) possibles.

Primer cal establir quin IAMatriu es compararà, si el de 3 accions (Imatge 24) o el de 6 accions (Imatge 29). Com es pot veure en sengles gràfiques, l'IAMatriu de 6 accions dóna millor resultat que el de 3 accions a igual població i noves generacions, per tant s'usarà l'IAMatriu de 6 accions (Imatge 26) per a comparar.

D'igual manera, l'IARL podria usar-se el de 256 estats (Imatge 43) o el de 4096 estats (Imatge 33). Com ja s'ha establert en el capítol Simulació del RL, usant 4096 estats s'aconsegueix un lleuger avantatge sobre el de 256.

La mitja de la puntuació per 500 passos de la gràfica de IAMatriu (Imatge 26) és 530. La mitja de la puntuació per 500 passos de la gràfica de IARL amb 4096 estats és 364. Aquesta mitja tant baixa es deu als períodes de desaprenentatge entre un coneixement anterior i un coneixement millor posterior.

Per tant, el navegador genètic IAMatriu resultant de 1000 generacions és més eficaç que el navegador per aprenentatge per reforç IARL a l'inici.

9. Conclusions i treball futur

En aquest apartat es resum i conclou la feina de la tesi i es plantegen noves fites.

9.1. Conclusions

Aquesta tesi s'ha desenvolupat al voltant d'obtenir un bon navegador per a robots netejadors. S'ha plantejat la manera de com fer que el robot sigui capaç de millorar i adaptar-se a un entorn, a priori, desconegut pel fabricant.

Durant el desenvolupament s'han creat diversos navegadors basats en intel·ligència artificial.

S'han establert diversos mecanismes per a la creació de navegadors simples, però que alhora, demostraven un bon rendiment i requerint poques generacions de cerca.

S'ha creat un controlador en base d'algorismes genètics, IAMatriu, a la qual se li ha creat un algorisme per a partir de les lectures de sensors que rep i la informació codificada dels seus pocs gens, prengui decisions d'accions. S'ha aconseguit que aquest controlador obtingui bons resultats amb poques generacions de recerca.

S'ha creat un controlador en base d'aprenentatge per reforç (IARL) amb exactament els mateixos inputs i outputs possibles que la IAMatriu. Aquest fet ha permès comparar-les en igualtat en base al rendiment per temps disponible.

Un desavantatge de l'aprenentatge per reforç és la falta d'imaginació dels algorismes per aprenentatge per refoç en trobar-se en circumstàncies poc comuns o no contemplades en el seu coneixement acumulat. Haver incorporat una aleatorietat acotada en l'algorisme del controlador creat (IARL) , permet assegurar que el robot serà capaç de tenir imaginació quan les circumstancies provoquin que el coneixement previ no resolgui un conflicte eventual.

Una altra particularitat de l'algorisme en base d'aprenentatge per reforç creada és que s'ha definit un mètode que permet evitar la fase prèvia d'entrenament: queda integrada en el funcionament normal. L'únic requisit d'aquest controlador és tenir un sistema que mesuri la brossa recollida.

Mentre un altre robot s'entrena a casa del fabricant, aquest s'entrena a casa del client.

9.2. Treball futur

En aquest apartat s'enumeren una serie de possibles continuacions a la feina realitzada.

9.2.1. Creació d'una interfície gràfica del simulador

Un primer projecte de continuació d'aquesta tesi seria la creació d'una interfície senzilla tipus formulari per a parametritzar les variables i iniciar la simulació.

Actualment el sistema és un projecte de consola que per a executar el simulador cal crear un programa amb totes les simulacions i engegar-lo. Creant el simulador amb el mateix esquema i classes en un projecte amb un formulari com a interfície augmentaria la utilitat i la facilitat d'ús.

Durant la tesi, es visualitzaven les simulacions representat-les en mode text (amb lletres i espais). Aquesta visualització seria ràpidament millorada usant un objecte de dibuix on poder dibuixar el(s) robot(s), parets i brutícia.

9.2.2. Modificació del simulador per a simular dos o més robots

Un segon projecte de continuació d'aquesta tesi seria l'adaptació del Simulador a més d'un robot.

El disseny del sistema està pensat per a aquesta adaptació. Només és necessari el disseny de una classe nova d'Obstacle que tingués longitud infinitesimal i estigués definida per un sol punt en en pla. Es modifica fent que aquesta classe d'Obstacle nova apunti a un objecte Robot i que vagi canviant la seva posició amb la posició del robot (simplement caldrà consultar la posició del robot al que apunta cada cop que hagi de realitzar un càlcul de col·lisió o detecció). Addicionalment, caldrà implementar que el nou Obstacle robot no calculi res sobre el robot al qual representa.

El Simulador està pensat per ser apuntat per de zero a infinits robots. Tenint cura de que cada robot crei un Obstacle de tipus robot en el Simulador al que apunta i que en la inicialització els robots no es solapin, el sistema estarà preparat per a simular tots els robots necessaris movent-se alhora.

9.2.3. Creació de una nova variant de la IAMatriu on les accions possibles són determinades per gens

Un tercer projecte de continuació d'aquesta tesi seria la creació de una variant del controlador IAMatriu.

En aquesta variant les accions possibles no estarien determinades per les accions contingudes en CadenaAccio sinó que uns gens de IAMatriu fossin els determinadors de quines accions són possibles. Donat que a s'ha vist a l'IAParametreParametre que aquesta característica millora el rendiment.

9.2.4. Cerca genètica dels paràmetres alfa, gamma, mida del sliding window i delta de l'algorisme per aprenentatge per reforç

Un quart projecte de continuació d'aquesta tesi seria la cerca dels paràmetres més adients de l'algorisme per aprenentatge de reforç a través d'un model genètic.

Durant la tesi s'ha efectuat una cerca exhaustiva dels millors paràmetres descrits a base de provar totes les seves combinacions (entre 2 i 5 valors possibles per a cada paràmetre) durant varies execucions i després comparar els millors paràmetres a base de comparar les puntuacions.

El projecte de cerca genètica permetria explorar moltes més combinacions, i el que és millor, valors no prefixats (tot dependria de com triar les funcions de reproducció i mutació). Així es podria arribar a determinar una sèrie de valors de paràmetres (sub)òptim no contemplat en la cerca feta durant la tesi.

9.2.5. Paral·lelització del comput

Finalment, un projecte continuador de la tesi seria l'adequació del Simulador al còmput multi-threading, usant les eines que la plataforma .NET ja té implementades.

Aquestes eines provenen de la llibreria System.Threading (la qual s'usa actualment a la crida de la funció Sleep que atura momentàniament l'execució d'ordres en la visualització per pantalla de l'execució).

L'adaptació del programa al multi-threading seria senzilla amb l'ús de mutex, en aquelles parts en conflicte en cas d'accessos simultanis.

Donat que el sistema és altament paral·lelitzable en els càlculs més costosos (per exemple, determinar per a cada Obstacle si hi ha col·lisió o no n'hi ha), l'execució en multi-threading en comptes del single-threading actual seria beneficiosa.

10. Bibliografia

[1] *Wikipedia* : http://www.wikipedia.org

[2] Francisco S. Melo. *RLSS09: Robot Learning Summer School Reinforcement Learning Lab Session* 2009

[3] CSI/ITESM . *Producto interno y ortogonalidad.* http://www.mty.itesm.mx/etie/deptos/m/ma00-130/lecturas/m130-08.pdf

[4] Robot aspirador inteligente ASTAN Modelo BL001A. Manual de instrucciones.

[5] Anykode http://www.anykode.com/

[6] Cyberbotics http://www.cyberbotics.com/webots

[7] Microsoft http://www.microsoft.com/robotics

[8] iRobot http://www.irobot.com/

[9] Hacking Roomba http://hackingroomba.com/

[10] iRobot Create Programmable Robot http://store.irobot.com/product/index.jsp?productId=2586252